Answer Set Solving in Practice

Torsten Schaub
University of Potsdam
torsten@cs.uni-potsdam.de

Systems: Overview

1. Potassco
2. gringo
3. clasp
4. clingo
5. clingcon
6. claspfolio
7. clavis

Potassco

Outline

Potassco

potassco.sourceforge.net

Potassco, the Potsdam Answer Set Solving Collection,
bundles tools for ASP developed at the University of Potsdam,
for instance:

- Grounder gringo, lingo, pyngo
- Solver clasp, claspfolio, claspar, aspeed
- Grounder+Solver Clingo, Clingcon, ROSoClingo
- Further Tools aspartame, asp{un}cud, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, quontroller, etc

- Benchmark repository asparagus.cs.uni-potsdam.de
- Teaching material potassco.sourceforge.net/teaching.html

Potassco

potassco.sourceforge.net

Potassco, the Potsdam Answer Set Solving Collection,
bundles tools for ASP developed at the University of Potsdam,
for instance:

- Grounder gringo, lingo, pyngo
- Solver clasp, claspfolio, claspar, aspeed
- Grounder+Solver Clingo, Clingcon, ROSoClingo
- Further Tools aspartame, asp{un}cud, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, quontroller, etc

- Benchmark repository asparagus.cs.uni-potsdam.de
- Teaching material potassco.sourceforge.net/teaching.html

Potassco

potassco.sourceforge.net

Potassco, the Potsdam Answer Set Solving Collection,
bundles tools for ASP developed at the University of Potsdam,
for instance:

- Grounder gringo, lingo, pyngo
- Solver clasp, claspfolio, claspar, aspeed
- Grounder+Solver Clingo, Clingcon, ROSoClingo
- Further Tools aspartame, asp{un}cud, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, quontroller, etc

- Benchmark repository `asparagus.cs.uni-potsdam.de`
- Teaching material `potassco.sourceforge.net/teaching.html`

Potassco

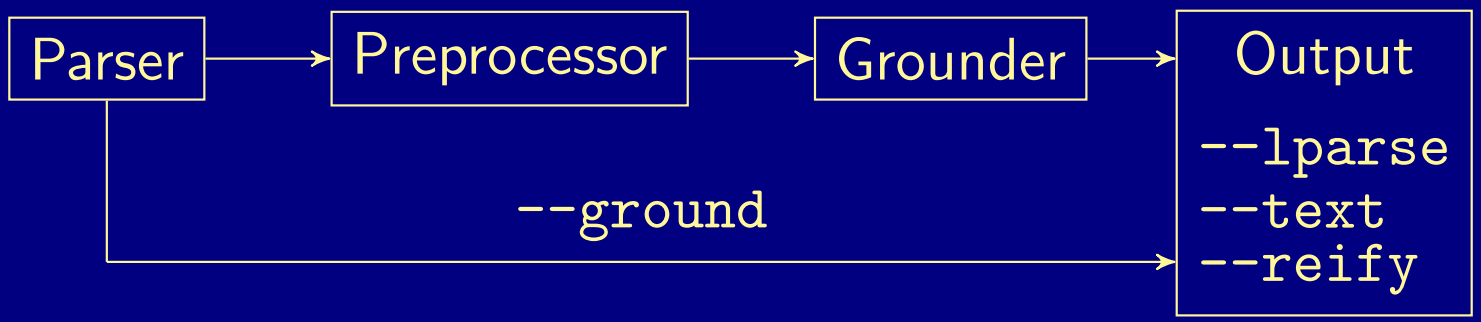
Outline

Potassco

gringo

- Accepts safe programs with aggregates
- Tolerates unrestricted use of function symbols
  (as long as it yields a finite ground instantiation :)
- Expressive power of a Turing machine

- Basic architecture of *gringo*:



Potassco

Outline

Potassco

Outline

Potassco

clasp

- *clasp* is a native ASP solver combining conflict-driven search with sophisticated reasoning techniques:
  - advanced preprocessing, including equivalence reasoning
  - lookback-based decision heuristics
  - restart policies
  - nogood deletion
  - progress saving
  - dedicated data structures for binary and ternary nogoods
  - lazy data structures (watched literals) for long nogoods
  - dedicated data structures for cardinality and weight constraints
  - lazy unfounded set checking based on "source pointers"
  - tight integration of unit propagation and unfounded set checking
  - various reasoning modes
  - parallel search
  - ...

Potassco

clasp

- *clasp* is a native ASP solver combining conflict-driven search with sophisticated reasoning techniques:
  - advanced preprocessing, including equivalence reasoning
  - lookback-based decision heuristics
  - restart policies
  - nogood deletion
  - progress saving
  - dedicated data structures for binary and ternary nogoods
  - lazy data structures (watched literals) for long nogoods
  - dedicated data structures for cardinality and weight constraints
  - lazy unfounded set checking based on "source pointers"
  - tight integration of unit propagation and unfounded set checking
  - various reasoning modes
  - parallel search
  - ...

Potassco

clasp

- *clasp* is a native ASP solver combining conflict-driven search with sophisticated reasoning techniques:
  - advanced preprocessing, including equivalence reasoning
  - lookback-based decision heuristics
  - restart policies
  - nogood deletion
  - progress saving
  - dedicated data structures for binary and ternary nogoods
  - lazy data structures (watched literals) for long nogoods
  - dedicated data structures for cardinality and weight constraints
  - lazy unfounded set checking based on "source pointers"
  - tight integration of unit propagation and unfounded set checking
  - various reasoning modes
  - parallel search
  - ...

Potassco

Reasoning modes of *clasp*

- Beyond deciding (stable) model existence, *clasp* allows for:
  - Optimization
  - Enumeration (without solution recording)
  - Projective enumeration (without solution recording)
  - Intersection and Union (linear solution computation)
  - and combinations thereof
- *clasp* allows for
  - ASP solving (*smodels* format)
  - MaxSAT and SAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)

Potassco

Reasoning modes of *clasp*

- Beyond deciding (stable) model existence, *clasp* allows for:
  - Optimization
  - Enumeration (without solution recording)
  - Projective enumeration (without solution recording)
  - Intersection and Union (linear solution computation)
  - and combinations thereof
- *clasp* allows for
  - ASP solving (*smodels* format)
  - MaxSAT and SAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)

Potassco

Outline

Potassco

Parallel search in *clasp*

- *clasp*
  - pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading
    - up to 64 configurable (non-hierarchic) threads
  - allows for parallel solving via search space splitting and/or competing strategies
    - both supported by solver portfolios
  - features different nogood exchange policies

Parallel search in *clasp*

- *clasp*
  - pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading
    - up to 64 configurable (non-hierarchic) threads
  - allows for parallel solving via search space splitting and/or competing strategies
    - both supported by solver portfolios
  - features different nogood exchange policies

Parallel search in *clasp*

- *clasp*
  - pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading
    - up to 64 configurable (non-hierarchic) threads
  - allows for parallel solving via search space splitting and/or competing strategies
    - both supported by solver portfolios
  - features different nogood exchange policies

Parallel search in *clasp*

- *clasp*
  - pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading
    - up to 64 configurable (non-hierarchic) threads
  - allows for parallel solving via search space splitting and/or competing strategies
    - both supported by solver portfolios
  - features different nogood exchange policies

Sequential CDCL-style solving

```
loop
    propagate                          // deterministically assign literals
    if no conflict then
        if all variables assigned then return solution
        else decide                    // non-deterministically assign some literal
    else
        if top-level conflict then return unsatisfiable
        else
            analyze                    // analyze conflict and add conflict constraint
            backjump                   // unassign literals until conflict constraint is unit
```

Potassco

Parallel CDCL-style solving in *clasp*

while work available
  while no (result) message to send
    communicate // exchange information with other solver
    propagate // deterministically assign literals
    if no conflict **then**
      if all variables assigned **then send** solution
      else *decide* // non-deterministically assign some literal
    else
      if root-level conflict **then send** unsatisfiable
      else if external conflict **then send** unsatisfiable
      else
        analyze // analyze conflict and add conflict constraint
        backjump // unassign literals until conflict constraint is unit
  communicate // exchange results (and receive work)

Potassco

Parallel CDCL-style solving in *clasp*

while work available
  while no (result) message to send
    communicate // exchange information with other solver
    propagate // deterministically assign literals
    if no conflict **then**
      if all variables assigned **then send** solution
      else *decide* // non-deterministically assign some literal
    else
      if root-level conflict **then send** unsatisfiable
      else if external conflict **then send** unsatisfiable
      else
        analyze // analyze conflict and add conflict constraint
        backjump // unassign literals until conflict constraint is unit
  communicate // exchange results (and receive work)

Potassco

Parallel CDCL-style solving in *clasp*

while work available
  while no (result) message to send
    communicate // exchange information with other solver
    propagate // deterministically assign literals
    if no conflict **then**
      if all variables assigned **then send** solution
      else *decide* // non-deterministically assign some literal
    else
      if root-level conflict **then send** unsatisfiable
      else if external conflict **then send** unsatisfiable
      else
        analyze // analyze conflict and add conflict constraint
        backjump // unassign literals until conflict constraint is unit
  communicate // exchange results (and receive work)

Potassco

Parallel CDCL-style solving in *clasp*

```
while work available
  while no (result) message to send
    communicate                    // exchange information with other solver
    propagate                      // deterministically assign literals
    if no conflict then
        if all variables assigned then send solution
        else decide                // non-deterministically assign some literal
    else
        if root-level conflict then send unsatisfiable
        else if external conflict then send unsatisfiable
        else
            analyze                // analyze conflict and add conflict constraint
            backjump               // unassign literals until conflict constraint is unit
  communicate                      // exchange results (and receive work)
```

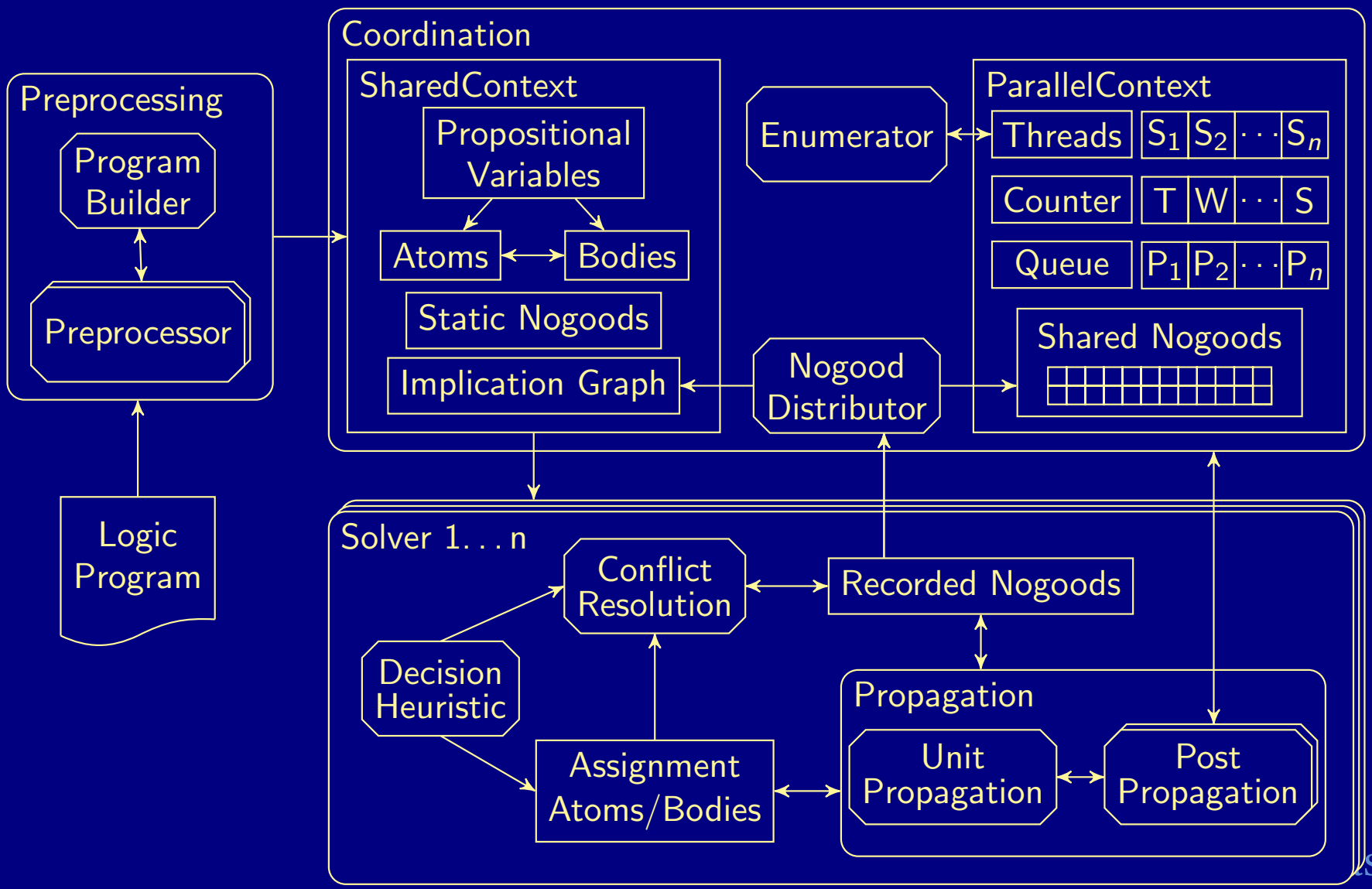
Potassco

Parallel CDCL-style solving in *clasp*

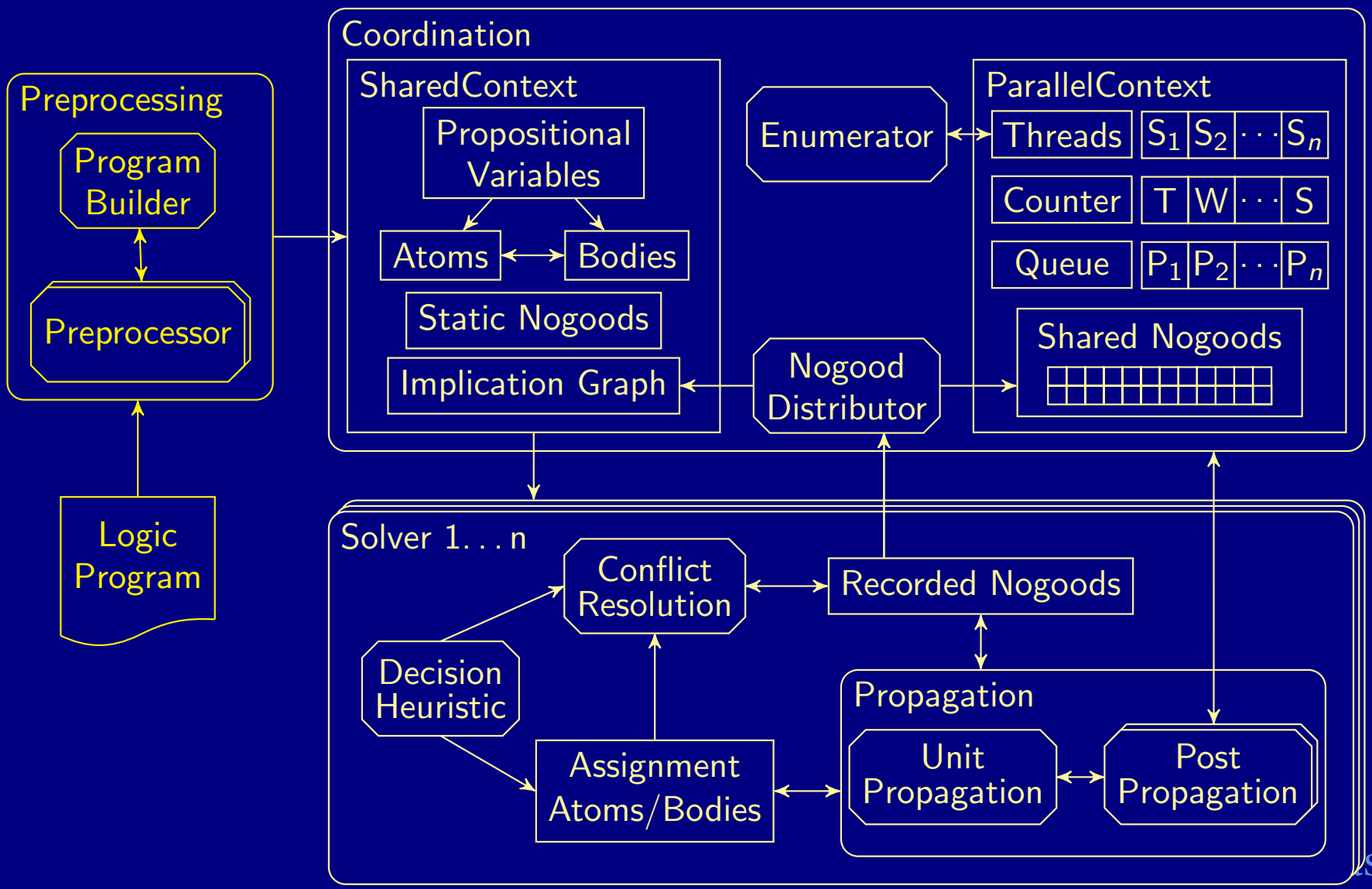
```
while work available
  while no (result) message to send
    communicate                    // exchange information with other solver
    propagate                      // deterministically assign literals
    if no conflict then
        if all variables assigned then send solution
        else decide                // non-deterministically assign some literal
    else
        if root-level conflict then send unsatisfiable
        else if external conflict then send unsatisfiable
        else
            analyze                // analyze conflict and add conflict constraint
            backjump               // unassign literals until conflict constraint is unit
  communicate                      // exchange results (and receive work)
```

Potassco

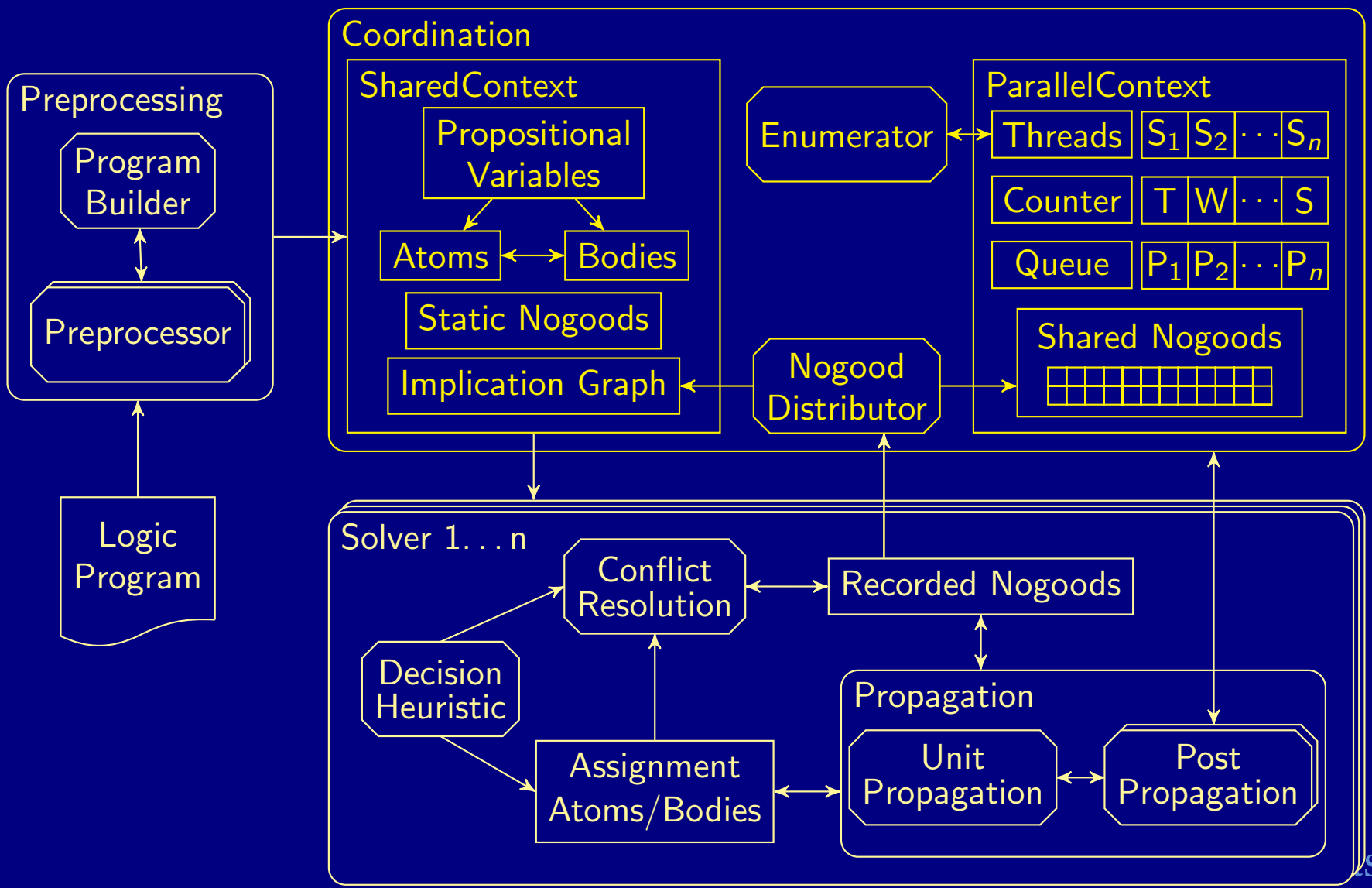
Multi-threaded architecture of *clasp*



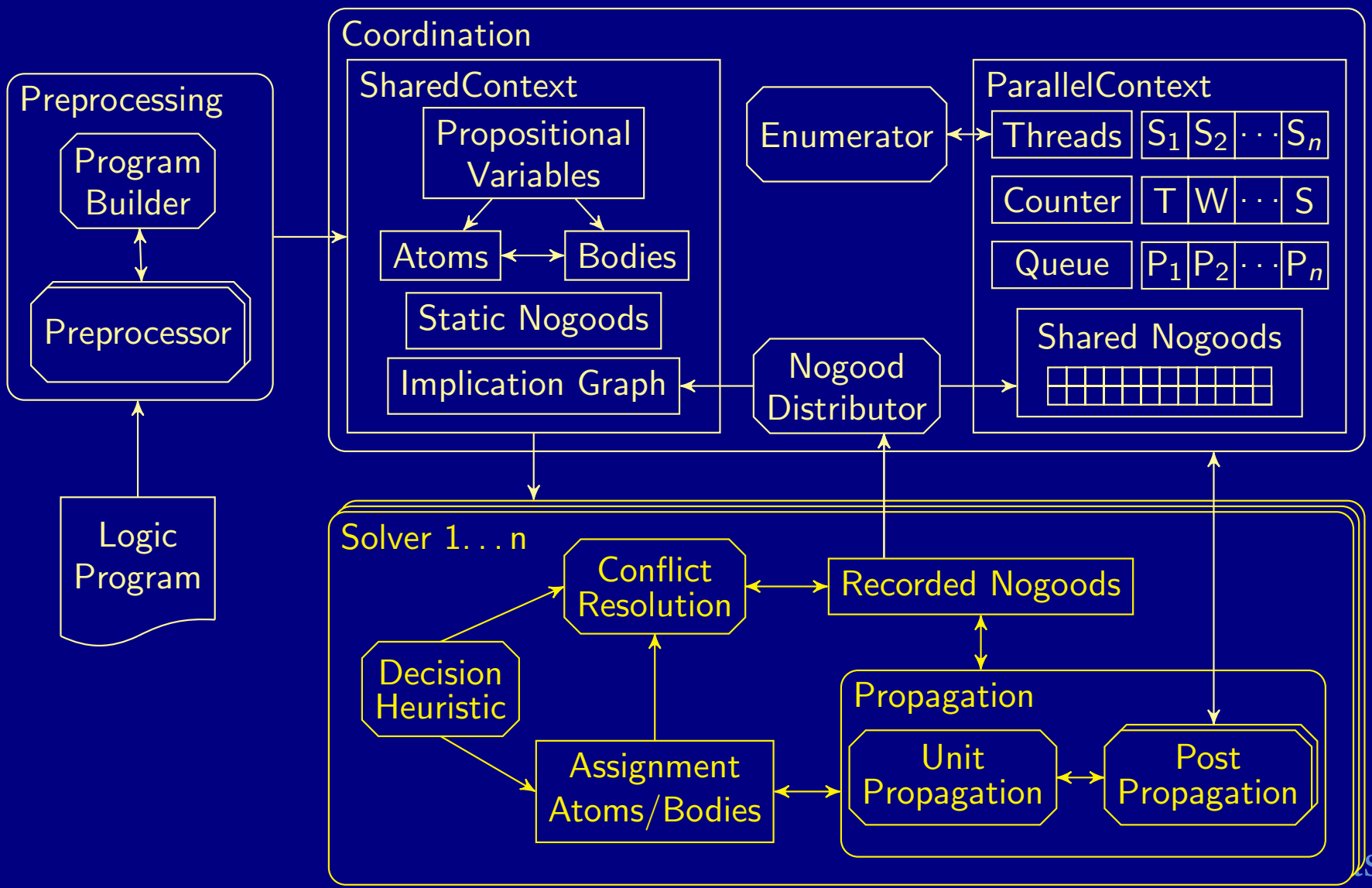
Multi-threaded architecture of *clasp*



Multi-threaded architecture of *clasp*



Multi-threaded architecture of *clasp*



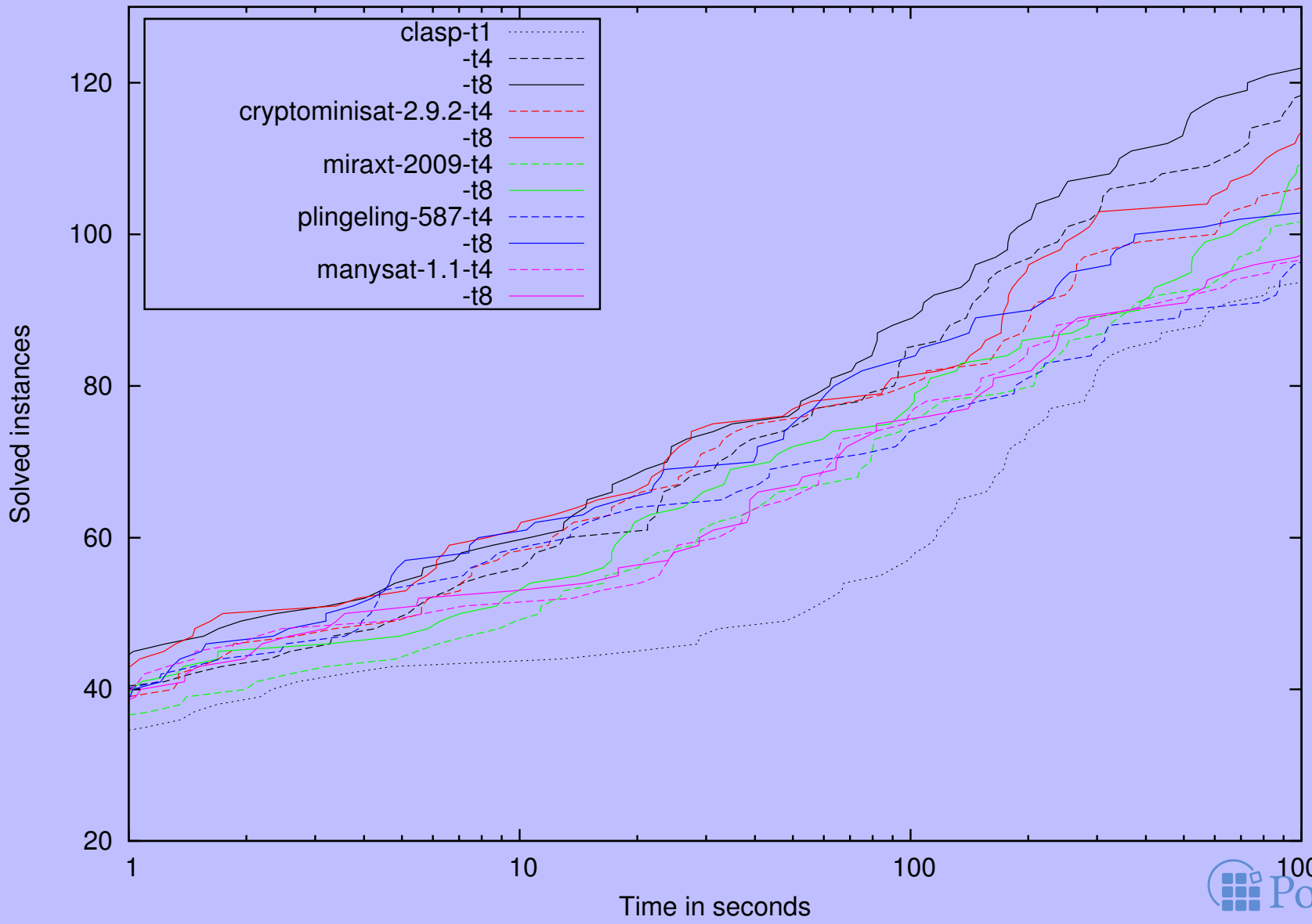
clasp in context

- Compare *clasp* (2.0.5) to the multi-threaded SAT solvers
  - *cryptominisat* (2.9.2)
  - *manysat* (1.1)
  - *miraxt* (2009)
  - *plingeling* (587f)

  all run with four and eight threads in their default settings
- 160/300 benchmarks from crafted category at SAT'11
  - all solvable by *ppfolio* in 1000 seconds
  - crafted SAT benchmarks are closest to ASP benchmarks

Potassco

clasp in context



Outline

Using *clasp*

```
--help[=<n>],-h          : Print {1=basic|2=more|3=full} help and exit

--parallel-mode,-t <arg>: Run parallel search with given number of threads
    <arg>: <n {1..64}>[,<mode {compete|split}>]
      <n>   : Number of threads to use in search
      <mode>: Run competition or splitting based search [compete]

--configuration=<arg>    : Configure default configuration [frumpy]
    <arg>: {frumpy|jumpy|handy|crafty|trendy|chatty}
      frumpy: Use conservative defaults
      jumpy : Use aggressive defaults
      handy : Use defaults geared towards large problems
      crafty: Use defaults geared towards crafted problems
      trendy: Use defaults geared towards industrial problems

      "-t 4": Use 4 competing threads initialized via the default portfolio

--print-portfolio,-g     : Print default portfolio and exit
```

Potassco

Using *clasp*

```
--help[=<n>],-h          : Print {1=basic|2=more|3=full} help and exit

--parallel-mode,-t <arg>: Run parallel search with given number of threads
    <arg>: <n {1..64}>[,<mode {compete|split}>]
      <n>   : Number of threads to use in search
      <mode>: Run competition or splitting based search [compete]

--configuration=<arg>    : Configure default configuration [frumpy]
    <arg>: {frumpy|jumpy|handy|crafty|trendy|chatty}
      frumpy: Use conservative defaults
      jumpy : Use aggressive defaults
      handy : Use defaults geared towards large problems
      crafty: Use defaults geared towards crafted problems
      trendy: Use defaults geared towards industrial problems

      "-t 4": Use 4 competing threads initialized via the default portfolio

--print-portfolio,-g     : Print default portfolio and exit
```

Potassco

Using *clasp*

```
--help[=<n>],-h          : Print {1=basic|2=more|3=full} help and exit

--parallel-mode,-t <arg>: Run parallel search with given number of threads
    <arg>: <n {1..64}>[,<mode {compete|split}>]
      <n>   : Number of threads to use in search
      <mode>: Run competition or splitting based search [compete]

--configuration=<arg>    : Configure default configuration [frumpy]
    <arg>: {frumpy|jumpy|handy|crafty|trendy|chatty}
      frumpy: Use conservative defaults
      jumpy : Use aggressive defaults
      handy : Use defaults geared towards large problems
      crafty: Use defaults geared towards crafted problems
      trendy: Use defaults geared towards industrial problems

      "-t 4": Use 4 competing threads initialized via the default portfolio

--print-portfolio,-g     : Print default portfolio and exit
```

Potassco

Using *clasp*

```
--help[=<n>],-h          : Print {1=basic|2=more|3=full} help and exit

--parallel-mode,-t <arg>: Run parallel search with given number of threads
    <arg>: <n {1..64}>[,<mode {compete|split}>]
      <n>   : Number of threads to use in search
      <mode>: Run competition or splitting based search [compete]

--configuration=<arg>    : Configure default configuration [frumpy]
    <arg>: {frumpy|jumpy|handy|crafty|trendy|chatty}
      frumpy: Use conservative defaults
      jumpy : Use aggressive defaults
      handy : Use defaults geared towards large problems
      crafty: Use defaults geared towards crafted problems
      trendy: Use defaults geared towards industrial problems

      "-t 4": Use 4 competing threads initialized via the default portfolio

--print-portfolio,-g     : Print default portfolio and exit
```

Potassco

Using *clasp*

```
--help[=<n>],-h          : Print {1=basic|2=more|3=full} help and exit

--parallel-mode,-t <arg>: Run parallel search with given number of threads
    <arg>: <n {1..64}>[,<mode {compete|split}>]
      <n>   : Number of threads to use in search
      <mode>: Run competition or splitting based search [compete]

--configuration=<arg>    : Configure default configuration [frumpy]
    <arg>: {frumpy|jumpy|handy|crafty|trendy|chatty}
      frumpy: Use conservative defaults
      jumpy : Use aggressive defaults
      handy : Use defaults geared towards large problems
      crafty: Use defaults geared towards crafted problems
      trendy: Use defaults geared towards industrial problems

      "-t 4": Use 4 competing threads initialized via the default portfolio

--print-portfolio,-g     : Print default portfolio and exit
```

Potassco

Comparing configurations

on `queensA.lp`

n	frumpy	jumpy	handy	crafty	trendy	-t 4
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Potassco

Comparing configurations

on queensA.lp

n	frumpy	jumpy	handy	crafty	trendy	-t 4
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Potassco

Comparing configurations

on queensA.lp

n	frumpy	jumpy	handy	crafty	trendy	-t 4
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Potassco

Comparing configurations

on queensA.lp

n	frumpy	jumpy	handy	crafty	trendy	-t 4
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Potassco

Comparing configurations

on queensA.lp

n	frumpy	jumpy	handy	crafty	trendy	-t 4
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Potassco

clasp's default portfolio for parallel solving

via `clasp --print-portfolio`

```
[solver.0]: --heuristic=Vsids,92 --restarts=L,60 --deletion=basic,50,0 --del-max=2000000 --del-estimate=1 --del
[solver.1]: --heuristic=Vsids --restarts=D,100,0.7 --deletion=basic,50,0 --del-init=3.0,500,19500 --del-grow=1.
[solver.2]: --heuristic=Berkmin --restarts=x,100,1.5 --deletion=basic,75 --del-init=3.0,200,40000 --del-max=400
[solver.3]: --restarts=x,128,1.5 --deletion=basic,75,0 --del-init=10.0,1000,9000 --del-grow=1.1,20.0 --del-cfl=
[solver.4]: --heuristic=Vsids --restarts=L,100 --deletion=basic,75,2 --del-init=3.0,1000,20000 --del-grow=1.1,2
[solver.5]: --heuristic=Vsids --restarts=D,100,0.7 --deletion=sort,50,2 --del-max=200000 --del-init=20.0,1000,1
[solver.6]: --heuristic=Berkmin,512 --restarts=x,100,1.5 --deletion=basic,75 --del-init=3.0,200,40000 --del-max
[solver.7]: --heuristic=Vsids --reverse-arcs=1 --otfs=1 --local-restarts --save-progress=0 --contraction=250 --
[solver.8]: --heuristic=Vsids --restarts=L,256 --counter-restart=3 --strengthen=recursive --update-lbd --del-gl
[solver.9]: --heuristic=Berkmin,512 --restarts=F,16000 --lookahead=atom,50
[solver.10]: --heuristic=Vmtf --strengthen=no --contr=0 --restarts=x,100,1.3 --del-init=3.0,800,9200
[solver.11]: --heuristic=Vsids --strengthen=recursive --restarts=x,100,1.5,15 --contraction=0
[solver.12]: --heuristic=Vsids --restarts=L,128 --save-p --otfs=1 --init-w=2 --contr=0 --opt-heu=3
[solver.13]: --heuristic=Berkmin,512 --restarts=x,100,1.5,6 --local-restarts --init-w=2 --contr=0
[solver.14]: --no-lookback --heuristic=Unit --lookahead=atom --deletion=no --restarts=no
```

- *clasp*'s portfolio is fully customizable
- configurations are assigned in a round-robin fashion to threads during parallel solving
- `-t 4` uses four threads with `crafty`, `trendy`, `frumpy`, and `jumpy`

clasp's default portfolio for parallel solving

via `clasp --print-portfolio`

```
[solver.0]: --heuristic=Vsids,92 --restarts=L,60 --deletion=basic,50,0 --del-max=2000000 --del-estimate=1 --del
[solver.1]: --heuristic=Vsids --restarts=D,100,0.7 --deletion=basic,50,0 --del-init=3.0,500,19500 --del-grow=1.
[solver.2]: --heuristic=Berkmin --restarts=x,100,1.5 --deletion=basic,75 --del-init=3.0,200,40000 --del-max=400
[solver.3]: --restarts=x,128,1.5 --deletion=basic,75,0 --del-init=10.0,1000,9000 --del-grow=1.1,20.0 --del-cfl=
[solver.4]: --heuristic=Vsids --restarts=L,100 --deletion=basic,75,2 --del-init=3.0,1000,20000 --del-grow=1.1,2
[solver.5]: --heuristic=Vsids --restarts=D,100,0.7 --deletion=sort,50,2 --del-max=200000 --del-init=20.0,1000,1
[solver.6]: --heuristic=Berkmin,512 --restarts=x,100,1.5 --deletion=basic,75 --del-init=3.0,200,40000 --del-max
[solver.7]: --heuristic=Vsids --reverse-arcs=1 --otfs=1 --local-restarts --save-progress=0 --contraction=250 --
[solver.8]: --heuristic=Vsids --restarts=L,256 --counter-restart=3 --strengthen=recursive --update-lbd --del-gl
[solver.9]: --heuristic=Berkmin,512 --restarts=F,16000 --lookahead=atom,50
[solver.10]: --heuristic=Vmtf --strengthen=no --contr=0 --restarts=x,100,1.3 --del-init=3.0,800,9200
[solver.11]: --heuristic=Vsids --strengthen=recursive --restarts=x,100,1.5,15 --contraction=0
[solver.12]: --heuristic=Vsids --restarts=L,128 --save-p --otfs=1 --init-w=2 --contr=0 --opt-heu=3
[solver.13]: --heuristic=Berkmin,512 --restarts=x,100,1.5,6 --local-restarts --init-w=2 --contr=0
[solver.14]: --no-lookback --heuristic=Unit --lookahead=atom --deletion=no --restarts=no
```

- *clasp*'s portfolio is fully customizable
- configurations are assigned in a round-robin fashion to threads during parallel solving
- `-t 4` uses four threads with `crafty`, `trendy`, `frumpy`, and `jumpy`

Potassco

clasp's default portfolio for parallel solving

via `clasp --print-portfolio`

```
[solver.0]: --heuristic=Vsids,92 --restarts=L,60 --deletion=basic,50,0 --del-max=2000000 --del-estimate=1 --del
[solver.1]: --heuristic=Vsids --restarts=D,100,0.7 --deletion=basic,50,0 --del-init=3.0,500,19500 --del-grow=1.
[solver.2]: --heuristic=Berkmin --restarts=x,100,1.5 --deletion=basic,75 --del-init=3.0,200,40000 --del-max=400
[solver.3]: --restarts=x,128,1.5 --deletion=basic,75,0 --del-init=10.0,1000,9000 --del-grow=1.1,20.0 --del-cfl=
[solver.4]: --heuristic=Vsids --restarts=L,100 --deletion=basic,75,2 --del-init=3.0,1000,20000 --del-grow=1.1,2
[solver.5]: --heuristic=Vsids --restarts=D,100,0.7 --deletion=sort,50,2 --del-max=200000 --del-init=20.0,1000,1
[solver.6]: --heuristic=Berkmin,512 --restarts=x,100,1.5 --deletion=basic,75 --del-init=3.0,200,40000 --del-max
[solver.7]: --heuristic=Vsids --reverse-arcs=1 --otfs=1 --local-restarts --save-progress=0 --contraction=250 --
[solver.8]: --heuristic=Vsids --restarts=L,256 --counter-restart=3 --strengthen=recursive --update-lbd --del-gl
[solver.9]: --heuristic=Berkmin,512 --restarts=F,16000 --lookahead=atom,50
[solver.10]: --heuristic=Vmtf --strengthen=no --contr=0 --restarts=x,100,1.3 --del-init=3.0,800,9200
[solver.11]: --heuristic=Vsids --strengthen=recursive --restarts=x,100,1.5,15 --contraction=0
[solver.12]: --heuristic=Vsids --restarts=L,128 --save-p --otfs=1 --init-w=2 --contr=0 --opt-heu=3
[solver.13]: --heuristic=Berkmin,512 --restarts=x,100,1.5,6 --local-restarts --init-w=2 --contr=0
[solver.14]: --no-lookback --heuristic=Unit --lookahead=atom --deletion=no --restarts=no
```

- *clasp*'s portfolio is fully customizable
- configurations are assigned in a round-robin fashion to threads during parallel solving
- `-t 4` uses four threads with `crafty`, `trendy`, `frumpy`, and `jumpy`

Outline

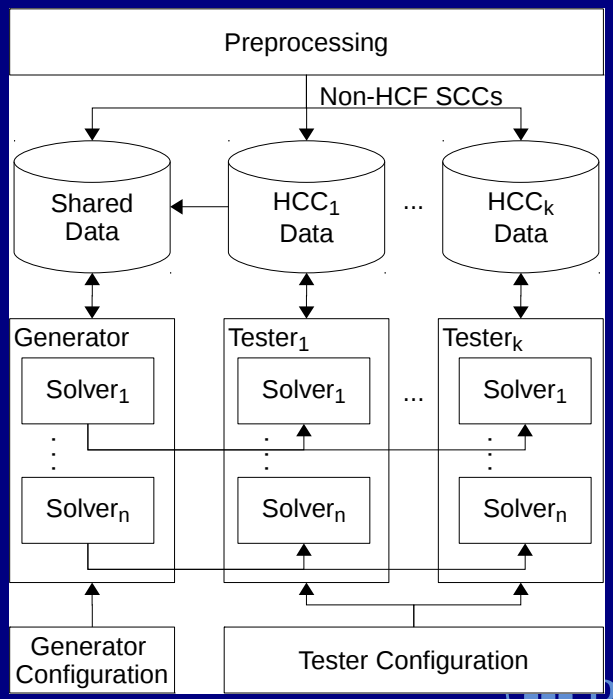
Potassco

clasp

- *clasp* is a multi-threaded solver for disjunctive logic programs
- aiming at an equitable interplay between "generating" and "testing" solver units
- allowing for a bidirectional dynamic information exchange between solver units for orthogonal tasks

clasp

- *clasp* is a multi-threaded solver for disjunctive logic programs
- aiming at an equitable interplay between "generating" and "testing" solver units
- allowing for a bidirectional dynamic information exchange between solver units for orthogonal tasks



Outline

hclasp

- *clasp* allows for incorporating domain-specific heuristics into ASP solving
  - input language for expressing domain-specific heuristics
  - solving capacities for integrating domain-specific heuristics
- Example

```
_heuristics(occ(A,T),factor,T) :- action(A), time(T).
```

hclasp

- *clasp* allows for incorporating domain-specific heuristics into ASP solving
  - input language for expressing domain-specific heuristics
  - solving capacities for integrating domain-specific heuristics
- Example

```
_heuristics(occ(A,T),factor,T) :- action(A), time(T).
```

Basic CDCL decision algorithm

loop

propagate // compute deterministic consequences

if no conflict **then**

if all variables assigned **then return** variable assignment

else decide // non-deterministically assign some literal

else

if top-level conflict **then return** unsatisfiable

else

analyze // analyze conflict and add a conflict constraint

backjump // undo assignments until conflict constraint is unit

Basic CDCL decision algorithm

loop

propagate // compute deterministic consequences

if no conflict **then**

if all variables assigned **then return** variable assignment

else decide // non-deterministically assign some literal

else

if top-level conflict **then return** unsatisfiable

else

analyze // analyze conflict and add a conflict constraint

backjump // undo assignments until conflict constraint is unit

Inside *decide*

- Heuristic functions

$$h : \mathcal{A} \to [0, +\infty) \quad \text{and} \quad s : \mathcal{A} \to \{\boldsymbol{T}, \boldsymbol{F}\}$$

- Algorithmic scheme

1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
2. $U := \mathcal{A} \setminus (A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}})$
3. $C := \mathit{argmax}_{a \in U} h(a)$
4. $a := \tau(C)$
5. $A := A \cup \{a \mapsto s(a)\}$

Potassco

Inside *decide*

- Heuristic functions

$$h : \mathcal{A} \to [0, +\infty) \quad \text{and} \quad s : \mathcal{A} \to \{\boldsymbol{T}, \boldsymbol{F}\}$$

- Algorithmic scheme

1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
2. $U := \mathcal{A} \setminus (A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}})$
3. $C := \mathit{argmax}_{a \in U} h(a)$
4. $a := \tau(C)$
5. $A := A \cup \{a \mapsto s(a)\}$

Potassco

Inside *decide*

- Heuristic functions

$$h : \mathcal{A} \to [0, +\infty) \quad \text{and} \quad s : \mathcal{A} \to \{\boldsymbol{T}, \boldsymbol{F}\}$$

- Algorithmic scheme

1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
2. $U := \mathcal{A} \setminus (A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}})$
3. $C := \mathit{argmax}_{a \in U} h(a)$
4. $a := \tau(C)$
5. $A := A \cup \{a \mapsto s(a)\}$

Potassco

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, *a*, and integer, *v*)

  `init` for initializing the heuristic value of *a* with *v*

  `factor` for amplifying the heuristic value of *a* by factor *v*

  `level` for ranking all atoms; the rank of *a* is *v*

  `sign` for attributing the sign of *v* as truth value to *a*

- Heuristic atoms

```
_heuristic(occurs(A,T),factor,T) :- action(A), time(T).
```

Potassco

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, a, and integer, v)

`init`	for initializing the heuristic value of a with v
`factor`	for amplifying the heuristic value of a by factor v
`level`	for ranking all atoms; the rank of a is v
`sign`	for attributing the sign of v as truth value to a

- Heuristic atoms

```
_heuristic(occurs(A,T),factor,T) :- action(A), time(T).
```

Potassco

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, a, and integer, v)

`init`	for initializing the heuristic value of a with v
`factor`	for amplifying the heuristic value of a by factor v
`level`	for ranking all atoms; the rank of a is v
`sign`	for attributing the sign of v as truth value to a

- Heuristic atoms

```
_heuristic(occurs(A,T),factor,T) :- action(A), time(T).
```

Potassco

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, a, and integer, v)

`init`	for initializing the heuristic value of a with v
`factor`	for amplifying the heuristic value of a by factor v
`level`	for ranking all atoms; the rank of a is v
`sign`	for attributing the sign of v as truth value to a

- Heuristic atoms

```
_heuristic(occurs(mv,5),factor,5) :- action(mv), time(5).
```

Potassco

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, a, and integer, v)

`init`	for initializing the heuristic value of a with v
`factor`	for amplifying the heuristic value of a by factor v
`level`	for ranking all atoms; the rank of a is v
`sign`	for attributing the sign of v as truth value to a

- Heuristic atoms

```
_heuristic(occurs(mv,5),factor,5) :- action(mv), time(5).
```

Potassco

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, a, and integer, v)

`init`	for initializing the heuristic value of a with v
`factor`	for amplifying the heuristic value of a by factor v
`level`	for ranking all atoms; the rank of a is v
`sign`	for attributing the sign of v as truth value to a

- Heuristic atoms

```
_heuristic(occurs(mv,5),factor,5) :- action(mv), time(5).
```

Potassco

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, *a*, and integer, *v*)

`init`	for initializing the heuristic value of *a* with *v*
`factor`	for amplifying the heuristic value of *a* by factor *v*
`level`	for ranking all atoms; the rank of *a* is *v*
`sign`	for attributing the sign of *v* as truth value to *a*

- Heuristic atoms

```
_heuristic(occurs(mv,5),factor,5) :- action(mv), time(5).
```

Potassco

Simple STRIPS planner

```
time(1..t).

holds(P,0) :- init(P).

1 { occurs(A,T) : action(A) } 1 :- time(T).
 :- occurs(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- holds(F,T-1), not nolds(F,T), time(T).
holds(F,T) :- occurs(A,T), add(A,F).
nolds(F,T) :- occurs(A,T), del(A,F).

 :- query(F), not holds(F,t).
```

Potassco

Simple STRIPS planner

```
time(1..t).

holds(P,0) :- init(P).

1 { occurs(A,T) : action(A) } 1 :- time(T).
 :- occurs(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- holds(F,T-1), not nolds(F,T), time(T).
holds(F,T) :- occurs(A,T), add(A,F).
nolds(F,T) :- occurs(A,T), del(A,F).

 :- query(F), not holds(F,t).

_heuristic(occurs(A,T),factor,2) :- action(A), time(T).
```

Potassco

Simple STRIPS planner

```
time(1..t).

holds(P,0) :- init(P).

1 { occurs(A,T) : action(A) } 1 :- time(T).
 :- occurs(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- holds(F,T-1), not nolds(F,T), time(T).
holds(F,T) :- occurs(A,T), add(A,F).
nolds(F,T) :- occurs(A,T), del(A,F).

 :- query(F), not holds(F,t).

_heuristic(occurs(A,T),level,1) :- action(A), time(T).
```

Simple STRIPS planner

```
time(1..t).

holds(P,0) :- init(P).

1 { occurs(A,T) : action(A) } 1 :- time(T).
 :- occurs(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- holds(F,T-1), not nolds(F,T), time(T).
holds(F,T) :- occurs(A,T), add(A,F).
nolds(F,T) :- occurs(A,T), del(A,F).

 :- query(F), not holds(F,t).

_heuristic(occurs(A,T),factor,T) :- action(A), time(T).
```

Simple STRIPS planner

```
time(1..t).

holds(P,0) :- init(P).

1 { occurs(A,T) : action(A) } 1 :- time(T).
 :- occurs(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- holds(F,T-1), not nolds(F,T), time(T).
holds(F,T) :- occurs(A,T), add(A,F).
nolds(F,T) :- occurs(A,T), del(A,F).

 :- query(F), not holds(F,t).

_heuristic(A,level,V) :- _heuristic(A,true, V).
_heuristic(A,sign, 1) :- _heuristic(A,true, V).
```

Potassco

Simple STRIPS planner

```
time(1..t).

holds(P,0) :- init(P).

1 { occurs(A,T) : action(A) } 1 :- time(T).
 :- occurs(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- holds(F,T-1), not nolds(F,T), time(T).
holds(F,T) :- occurs(A,T), add(A,F).
nolds(F,T) :- occurs(A,T), del(A,F).

 :- query(F), not holds(F,t).

_heuristic(A,level,V) :- _heuristic(A,false,V).
_heuristic(A,sign,-1) :- _heuristic(A,false,V).
```

Potassco

Simple STRIPS planner

```
time(1..t).

holds(P,0) :- init(P).

1 { occurs(A,T) : action(A) } 1 :- time(T).
 :- occurs(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- holds(F,T-1), not nolds(F,T), time(T).
holds(F,T) :- occurs(A,T), add(A,F).
nolds(F,T) :- occurs(A,T), del(A,F).

 :- query(F), not holds(F,t).

_heuristic(holds(F,T-1),true, t-T+1) :- holds(F,T).
_heuristic(holds(F,T-1),false,t-T+1) :-
                fluent(F), time(T), not holds(F,T).
```

Heuristic modifications to functions h and s

- $\nu(V_{a,m}(A))$ — *"value for modifier m on atom a wrt assignment A"*
- `init` and

$$d_0(a) = \nu(V_{a,\texttt{init}}(A_0)) + h_0(a)$$

$$d_i(a) = \begin{cases} \nu(V_{a,\texttt{factor}}(A_i)) \times h_i(a) & \text{if } V_{a,\texttt{factor}}(A_i) \neq \emptyset \\ h_i(a) & \text{otherwise} \end{cases}$$

- `sign`

$$t_i(a) = \begin{cases} \boldsymbol{T} & \text{if } \nu(V_{a,\texttt{sign}}(A_i)) > 0 \\ \boldsymbol{F} & \text{if } \nu(V_{a,\texttt{sign}}(A_i)) < 0 \\ s_i(a) & \text{otherwise} \end{cases}$$

- `level` $\ell_{A_i}(\mathcal{A}') = \mathit{argmax}_{a \in \mathcal{A}'} \nu(V_{a,\texttt{level}}(A_i)) \qquad \mathcal{A}' \subseteq \mathcal{A}$

Heuristic modifications to functions h and s

- $\nu(V_{a,m}(A))$ — "*value for modifier m on atom a wrt assignment A*"
- `init` and `factor`

$$d_0(a) = \nu(V_{a,\texttt{init}}(A_0)) + h_0(a)$$

$$d_i(a) = \begin{cases} \nu(V_{a,\texttt{factor}}(A_i)) \times h_i(a) & \text{if } V_{a,\texttt{factor}}(A_i) \neq \emptyset \\ h_i(a) & \text{otherwise} \end{cases}$$

- `sign`

$$t_i(a) = \begin{cases} \boldsymbol{T} & \text{if } \nu(V_{a,\texttt{sign}}(A_i)) > 0 \\ \boldsymbol{F} & \text{if } \nu(V_{a,\texttt{sign}}(A_i)) < 0 \\ s_i(a) & \text{otherwise} \end{cases}$$

- `level` $\ell_{A_i}(\mathcal{A}') = \mathit{argmax}_{a \in \mathcal{A}'} \nu(V_{a,\texttt{level}}(A_i)) \qquad \mathcal{A}' \subseteq \mathcal{A}$

Heuristic modifications to functions h and s

- $\nu(V_{a,m}(A))$ — *"value for modifier m on atom a wrt assignment A"*
- `init` and `factor`

$$d_0(a) = \nu(V_{a,\texttt{init}}(A_0)) + h_0(a)$$

$$d_i(a) = \begin{cases} \nu(V_{a,\texttt{factor}}(A_i)) \times h_i(a) & \text{if } V_{a,\texttt{factor}}(A_i) \neq \emptyset \\ h_i(a) & \text{otherwise} \end{cases}$$

- `sign`

$$t_i(a) = \begin{cases} \boldsymbol{T} & \text{if } \nu(V_{a,\texttt{sign}}(A_i)) > 0 \\ \boldsymbol{F} & \text{if } \nu(V_{a,\texttt{sign}}(A_i)) < 0 \\ s_i(a) & \text{otherwise} \end{cases}$$

- `level` $\ell_{A_i}(\mathcal{A}') = \mathit{argmax}_{a \in \mathcal{A}'} \nu(V_{a,\texttt{level}}(A_i)) \qquad \mathcal{A}' \subseteq \mathcal{A}$

Heuristic modifications to functions h and s

- $\nu(V_{a,m}(A))$ — *"value for modifier m on atom a wrt assignment A"*
- `init` and `factor`

$$d_0(a) = \nu(V_{a,\texttt{init}}(A_0)) + h_0(a)$$

$$d_i(a) = \begin{cases} \nu(V_{a,\texttt{factor}}(A_i)) \times h_i(a) & \text{if } V_{a,\texttt{factor}}(A_i) \neq \emptyset \\ h_i(a) & \text{otherwise} \end{cases}$$

- `sign`

$$t_i(a) = \begin{cases} \boldsymbol{T} & \text{if } \nu(V_{a,\texttt{sign}}(A_i)) > 0 \\ \boldsymbol{F} & \text{if } \nu(V_{a,\texttt{sign}}(A_i)) < 0 \\ s_i(a) & \text{otherwise} \end{cases}$$

- `level` $\ell_{A_i}(\mathcal{A}') = \mathit{argmax}_{a \in \mathcal{A}'} \nu(V_{a,\texttt{level}}(A_i)) \qquad \mathcal{A}' \subseteq \mathcal{A}$

Heuristic modifications to functions h and s

- $\nu(V_{a,m}(A))$ — *"value for modifier m on atom a wrt assignment A"*
- `init` and `factor`

$$d_0(a) = \nu(V_{a,\texttt{init}}(A_0)) + h_0(a)$$

$$d_i(a) = \begin{cases} \nu(V_{a,\texttt{factor}}(A_i)) \times h_i(a) & \text{if } V_{a,\texttt{factor}}(A_i) \neq \emptyset \\ h_i(a) & \text{otherwise} \end{cases}$$

- `sign`

$$t_i(a) = \begin{cases} \boldsymbol{T} & \text{if } \nu(V_{a,\texttt{sign}}(A_i)) > 0 \\ \boldsymbol{F} & \text{if } \nu(V_{a,\texttt{sign}}(A_i)) < 0 \\ s_i(a) & \text{otherwise} \end{cases}$$

- `level` $\ell_{A_i}(\mathcal{A}') = \mathit{argmax}_{a \in \mathcal{A}'} \nu(V_{a,\texttt{level}}(A_i)) \qquad \mathcal{A}' \subseteq \mathcal{A}$

Potassco

Heuristic modifications to functions h and s

- $\nu(V_{a,m}(A))$ — "*value for modifier m on atom a wrt assignment A*"
- `init` and

$$d_0(a) = \nu(V_{a,\texttt{init}}(A_0)) + h_0(a)$$

$$d_i(a) = \begin{cases} \nu(V_{a,\texttt{factor}}(A_i)) \times h_i(a) & \text{if } V_{a,\texttt{factor}}(A_i) \neq \emptyset \\ h_i(a) & \text{otherwise} \end{cases}$$

- `sign`

$$t_i(a) = \begin{cases} \boldsymbol{T} & \text{if } \nu(V_{a,\texttt{sign}}(A_i)) > 0 \\ \boldsymbol{F} & \text{if } \nu(V_{a,\texttt{sign}}(A_i)) < 0 \\ s_i(a) & \text{otherwise} \end{cases}$$

- `level` $\ell_{A_i}(\mathcal{A}') = \mathit{argmax}_{a \in \mathcal{A}'} \nu(V_{a,\texttt{level}}(A_i))$ $\mathcal{A}' \subseteq \mathcal{A}$

Heuristic modifications to functions h and s

- $\nu(V_{a,m}(A))$ — "*value for modifier m on atom a wrt assignment A*"
- `init` and `factor`

$$d_0(a) = \nu(V_{a,\texttt{init}}(A_0)) + h_0(a)$$

$$d_i(a) = \begin{cases} \nu(V_{a,\texttt{factor}}(A_i)) \times h_i(a) & \text{if } V_{a,\texttt{factor}}(A_i) \neq \emptyset \\ h_i(a) & \text{otherwise} \end{cases}$$

- `sign`

$$t_i(a) = \begin{cases} \boldsymbol{T} & \text{if } \nu(V_{a,\texttt{sign}}(A_i)) > 0 \\ \boldsymbol{F} & \text{if } \nu(V_{a,\texttt{sign}}(A_i)) < 0 \\ s_i(a) & \text{otherwise} \end{cases}$$

- `level` $\ell_{A_i}(\mathcal{A}') = \mathit{argmax}_{a \in \mathcal{A}'} \nu(V_{a,\texttt{level}}(A_i)) \qquad \mathcal{A}' \subseteq \mathcal{A}$

Inside *decide*, heuristically modified

0. $h(a) := d(a)$ for each $a \in \mathcal{A}$
1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
2. $U := \ell_A(\mathcal{A} \setminus (A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}}))$
3. $C := argmax_{a \in U} d(a)$
4. $a := \tau(C)$
5. $A := A \cup \{a \mapsto t(a)\}$

Potassco

Inside *decide*, heuristically modified

0. $h(a) := d(a)$ for each $a \in \mathcal{A}$
1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
2. $U := \ell_A(\mathcal{A} \setminus (A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}}))$
3. $C := argmax_{a \in U} d(a)$
4. $a := \tau(C)$
5. $A := A \cup \{a \mapsto t(a)\}$

Inside *decide*, heuristically modified

0. $h(a) := d(a)$ for each $a \in \mathcal{A}$
1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
2. $U := \ell_A(\mathcal{A} \setminus (A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}}))$
3. $C := argmax_{a \in U} d(a)$
4. $a := \tau(C)$
5. $A := A \cup \{a \mapsto t(a)\}$

Selected high scores from systematic experiments

Setting	*Labyrinth*	*Sokoban*	*Hanoi Tower*
base configuration	9,108*s* (14) 24,545,667	2,844*s* (3) 19,371,267	9,137*s* (11) 41,016,235
a, `init`, 2	95% (12) 94%	91% (1) 84%	85% (9) 89%
a, `factor`, 4	78% (8) 30%	120% (1) 107%	109% (11) 110%
a, `factor`, 16	78% (10) 23%	120% (1) 107%	109% (11) 110%
a, `level`, 1	90% (12) 5%	119% (2) 91%	126% (15) 120%
f, `init`, 2	103% (14) 123%	74% (2) 71%	97% (10) 109%
f, `factor`, 2	98% (12) 49%	116% (3) 134%	55% (6) 70%
f, `sign`, `-1`	94% (13) 89%	105% (1) 100%	92% (12) 92%

base configuration versus 38 (static) heuristic modifications (action, a, and fluent, f)

Potassco

Selected high scores from systematic experiments

Setting	*Labyrinth*	*Sokoban*	*Hanoi Tower*
base configuration	9,108*s* (14) 24,545,667	2,844*s* (3) 19,371,267	9,137*s* (11) 41,016,235
a, `init`, 2	95% (12) 94%	91% (1) 84%	85% (9) 89%
a, `factor`, 4	78% (8) 30%	120% (1) 107%	109% (11) 110%
a, `factor`, 16	78% (10) 23%	120% (1) 107%	109% (11) 110%
a, `level`, 1	90% (12) 5%	119% (2) 91%	126% (15) 120%
f, `init`, 2	103% (14) 123%	74% (2) 71%	97% (10) 109%
f, `factor`, 2	98% (12) 49%	116% (3) 134%	55% (6) 70%
f, `sign`, `-1`	94% (13) 89%	105% (1) 100%	92% (12) 92%

base configuration versus 38 (static) heuristic modifications (action, a, and fluent, f)

Potassco

Abductive problems with optimization

Setting	*Diagnosis*	*Expansion*	*Repair (H)*	*Repair (S)*
base configuration	111.1*s* (115)	161.5*s* (100)	101.3*s* (113)	33.3*s* (27)
`sign,-1`	324.5*s* (407)	7.6*s* (3)	8.4*s* (5)	3.1*s* (0)
`sign,-1 factor,2`	310.1*s* (387)	7.4*s* (2)	3.5*s* (0)	3.2*s* (1)
`sign,-1 factor,8`	305.9*s* (376)	7.7*s* (2)	3.1*s* (0)	2.9*s* (0)
`sign,-1 level,1`	76.1*s* (83)	6.6*s* (2)	0.8*s* (0)	2.2*s* (1)
`level,1`	77.3*s* (86)	12.9*s* (5)	3.4*s* (0)	2.1*s* (0)

(abducibles subject to optimization)

Potassco

Abductive problems with optimization

Setting	*Diagnosis*	*Expansion*	*Repair (H)*	*Repair (S)*
base configuration	111.1*s* (115)	161.5*s* (100)	101.3*s* (113)	33.3*s* (27)
`sign,-1`	324.5*s* (407)	7.6*s* (3)	8.4*s* (5)	3.1*s* (0)
`sign,-1 factor,2`	310.1*s* (387)	7.4*s* (2)	3.5*s* (0)	3.2*s* (1)
`sign,-1 factor,8`	305.9*s* (376)	7.7*s* (2)	3.1*s* (0)	2.9*s* (0)
`sign,-1 level,1`	76.1*s* (83)	6.6*s* (2)	0.8*s* (0)	2.2*s* (1)
`level,1`	77.3*s* (86)	12.9*s* (5)	3.4*s* (0)	2.1*s* (0)

(abducibles subject to optimization)

Potassco

Planning Competition Benchmarks

```
_heuristic(holds(F,T-1),true, t-T+1) :- holds(F,T).
_heuristic(holds(F,T-1),false,t-T+1) :-
                  fluent(F), time(T), not holds(F,T).
```

Problem	*base configuration*		_heuristic		*base c. (SAT)*		_heur. *(SAT)*	
Blocks'00	134.4*s*	(180/61)	9.2*s*	(239/3)	163.2*s*	(59)	2.6*s*	(0)
Elevator'00	3.1*s*	(279/0)	0.0*s*	(279/0)	3.4*s*	(0)	0.0*s*	(0)
Freecell'00	288.7*s*	(147/115)	184.2*s*	(194/74)	226.4*s*	(47)	52.0*s*	(0)
Logistics'00	145.8*s*	(148/61)	115.3*s*	(168/52)	113.9*s*	(23)	15.5*s*	(3)
Depots'02	400.3*s*	(51/184)	297.4*s*	(115/135)	389.0*s*	(64)	61.6*s*	(0)
Driverlog'02	308.3*s*	(108/143)	189.6*s*	(169/92)	245.8*s*	(61)	6.1*s*	(0)
Rovers'02	245.8*s*	(138/112)	165.7*s*	(179/79)	162.9*s*	(41)	5.7*s*	(0)
Satellite'02	398.4*s*	(73/186)	229.9*s*	(155/106)	364.6*s*	(82)	30.8*s*	(0)
Zenotravel'02	350.7*s*	(101/169)	239.0*s*	(154/116)	224.5*s*	(53)	6.3*s*	(0)
Total	252.8*s*	(1225/1031)	158.9*s*	(1652/657)	187.2*s*	(430)	17.1*s*	(3)

Potassco

Planning Competition Benchmarks

```
_heuristic(holds(F,T-1),true, t-T+1) :- holds(F,T).
_heuristic(holds(F,T-1),false,t-T+1) :-
                  fluent(F), time(T), not holds(F,T).
```

Problem	*base configuration*		_heuristic		*base c. (SAT)*		_heur. *(SAT)*	
Blocks'00	134.4*s*	(180/61)	9.2*s*	(239/3)	163.2*s*	(59)	2.6*s*	(0)
Elevator'00	3.1*s*	(279/0)	0.0*s*	(279/0)	3.4*s*	(0)	0.0*s*	(0)
Freecell'00	288.7*s*	(147/115)	184.2*s*	(194/74)	226.4*s*	(47)	52.0*s*	(0)
Logistics'00	145.8*s*	(148/61)	115.3*s*	(168/52)	113.9*s*	(23)	15.5*s*	(3)
Depots'02	400.3*s*	(51/184)	297.4*s*	(115/135)	389.0*s*	(64)	61.6*s*	(0)
Driverlog'02	308.3*s*	(108/143)	189.6*s*	(169/92)	245.8*s*	(61)	6.1*s*	(0)
Rovers'02	245.8*s*	(138/112)	165.7*s*	(179/79)	162.9*s*	(41)	5.7*s*	(0)
Satellite'02	398.4*s*	(73/186)	229.9*s*	(155/106)	364.6*s*	(82)	30.8*s*	(0)
Zenotravel'02	350.7*s*	(101/169)	239.0*s*	(154/116)	224.5*s*	(53)	6.3*s*	(0)
Total	252.8*s*	(1225/1031)	158.9*s*	(1652/657)	187.2*s*	(430)	17.1*s*	(3)

Potassco

Planning Competition Benchmarks

```
_heuristic(holds(F,T-1),true, t-T+1) :- holds(F,T).
_heuristic(holds(F,T-1),false,t-T+1) :-
                  fluent(F), time(T), not holds(F,T).
```

Problem	*base configuration*		_heuristic		*base c. (SAT)*		_heur. *(SAT)*	
Blocks'00	134.4*s*	(180/61)	9.2*s*	(239/3)	163.2*s*	(59)	2.6*s*	(0)
Elevator'00	3.1*s*	(279/0)	0.0*s*	(279/0)	3.4*s*	(0)	0.0*s*	(0)
Freecell'00	288.7*s*	(147/115)	184.2*s*	(194/74)	226.4*s*	(47)	52.0*s*	(0)
Logistics'00	145.8*s*	(148/61)	115.3*s*	(168/52)	113.9*s*	(23)	15.5*s*	(3)
Depots'02	400.3*s*	(51/184)	297.4*s*	(115/135)	389.0*s*	(64)	61.6*s*	(0)
Driverlog'02	308.3*s*	(108/143)	189.6*s*	(169/92)	245.8*s*	(61)	6.1*s*	(0)
Rovers'02	245.8*s*	(138/112)	165.7*s*	(179/79)	162.9*s*	(41)	5.7*s*	(0)
Satellite'02	398.4*s*	(73/186)	229.9*s*	(155/106)	364.6*s*	(82)	30.8*s*	(0)
Zenotravel'02	350.7*s*	(101/169)	239.0*s*	(154/116)	224.5*s*	(53)	6.3*s*	(0)
Total	252.8*s*	(1225/1031)	158.9*s*	(1652/657)	187.2*s*	(430)	17.1*s*	(3)

Potassco

Outline

Potassco

Clingo species

Clingo = *gringo* | *clasp*

- Before
  - *Clingo* — easy solving
    iClingo — incremental solving
    oClingo — reactive solving
- After
  - *Clingo* — easy solving
    + incremental solving
    + reactive solving
    + complex solving
- *Clingo* series 4 = ASP + Control
- Multi-shot ASP solving deals with continously changing programs

Potassco

Clingo species

- Before

  Clingo = *gringo* | *clasp*

  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving

- After

  Clingo — easy solving
  + incremental solving
  + reactive solving
  + complex solving

- *Clingo* series 4 = ASP + Control
- Multi-shot ASP solving deals with continously changing programs

Potassco

Clingo species

Clingo = *gringo* | *clasp*

- Before
  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After

  Clingo — easy solving
  + incremental solving
  + reactive solving
  + complex solving

- *Clingo* series 4 = ASP + Control
- Multi-shot ASP solving deals with continously changing programs

Potassco

Clingo species

Clingo = *gringo* | *clasp*

- Before
  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After
  - *Clingo* — easy solving
    + incremental solving
    + reactive solving
    + complex solving
- *Clingo* series 4 = ASP + Control
- Multi-shot ASP solving deals with continously changing programs

Potassco

Clingo species

Clingo = *gringo* | *clasp*

- Before
  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After
  - *Clingo* — easy solving
  - + incremental solving
  - + reactive solving
  - + complex solving
- *Clingo* series 4 = ASP + Control
- Multi-shot ASP solving deals with continously changing programs

Potassco

Clingo species

- Before

  $Clingo = gringo \mid clasp$

  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving

- After

  $Clingo = gringo \mid clasp$

  - *Clingo* — easy solving
  - + incremental solving
  - + reactive solving
  - + complex solving

- *Clingo* series 4 = ASP + Control
- Multi-shot ASP solving deals with continously changing programs

Potassco

Clingo species

- Before

  $Clingo = gringo \mid clasp$

  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After

  $Clingo = gringo^* \mid clasp^*$

  - *Clingo* — easy solving
  - + incremental solving
  - + reactive solving
  - + complex solving
- *Clingo* series 4 = ASP + Control
- Multi-shot ASP solving deals with continously changing programs

Potassco

Clingo species

- Before $\quad Clingo \;=\; gringo \mid clasp$
  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After $\quad Clingo \;=\; (gringo^* \mid clasp^*)^*$
  - *Clingo* — easy solving
  - $+$ incremental solving
  - $+$ reactive solving
  - $+$ complex solving
- *Clingo* series 4 $=$ ASP $+$ Control
- Multi-shot ASP solving deals with continously changing programs

Potassco

Clingo species

- Before $\quad Clingo \;=\; gringo \mid clasp$
  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After $\quad Clingo \;=\; (gringo^* \mid clasp^*)^*$
  - *Clingo* — easy solving
  - $+$ incremental solving
  - $+$ reactive solving
  - $+$ complex solving
- *Clingo* series 4 $=$ ASP $+$ Control
- Multi-shot ASP solving deals with continously changing programs

Potassco

Clingo species

- Before $\quad Clingo \;=\; gringo \mid clasp$
  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After $\quad Clingo \;=\; (gringo^* \mid clasp^*)^*$
  - *Clingo* — easy solving
  - $+$ incremental solving
  - $+$ reactive solving
  - $+$ complex solving
- *Clingo* series 4 $=$ ASP $+$ Control
- Multi-shot ASP solving deals with continously changing programs

Potassco

Clingo species

- Before $\quad Clingo \;=\; gringo \mid clasp$
  - *Clingo* — easy solving
  - *iClingo* — incremental solving
  - *oClingo* — reactive solving
- After $\quad Clingo \;=\; (gringo^* \mid clasp^*)^*$
  - *Clingo* — easy solving
  - $+$ incremental solving
  - $+$ reactive solving
  - $+$ complex solving
- *Clingo* series 4 $\;=\;$ ASP $+$ Control
- Multi-shot ASP solving deals with continously changing programs

Potassco

Outline

Potassco

Clingo series 4

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
    - Default `#program base.`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
  - Grounding yields a pair (P, E) of ground rules and ground atoms, whose semantics is captured by module theory
- Control
  - Lua
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python
    - Example `prg.solve(), prg.ground(parts), ...`

  embedded scripting language (`#script`)
  libraries

Potassco

Clingo series 4

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
    - Default `#program base.`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
  - Grounding yields a pair (P, E) of ground rules and ground atoms, whose semantics is captured by module theory
- Control
  - Lua
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python
    - Example `prg.solve(), prg.ground(parts), ...`

  embedded scripting language (`#script`)
  libraries

Potassco

Clingo series 4

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
    - Default `#program base.`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
  - Grounding yields a pair (P, E) of ground rules and ground atoms, whose semantics is captured by module theory
- Control
  - Lua
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python
    - Example `prg.solve(), prg.ground(parts), ...`

  embedded scripting language (`#script`)

  libraries

Potassco

Clingo series 4

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
    - Default `#program base.`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
  - Grounding yields a pair (P, E) of ground rules and ground atoms, whose semantics is captured by module theory
- Control
  - Lua
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python
    - Example `prg.solve(), prg.ground(parts), ...`
  - embedded scripting language (`#script`)
  - libraries

Potassco

Clingo series 4

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
    - Default `#program base.`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
  - Grounding yields a pair (P, E) of ground rules and ground atoms, whose semantics is captured by module theory
- Control
  - Lua
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python
    - Example `prg.solve(), prg.ground(parts), ...`
  - embedded scripting language (`#script`)
  - libraries

Potassco

Vanilla *Clingo*

- Emulating *Clingo* in *Clingo* 4

```
#script (python)
def main(prg):
    parts = []
    parts.append(("base", []))
    prg.ground(parts)
    prg.solve()
#end.
```

Potassco

Vanilla *Clingo*

- Emulating *Clingo* in *Clingo* 4

```
#script (python)
def main(prg):
    parts = []
    parts.append(("base", []))
    prg.ground(parts)
    prg.solve()
#end.
```

Vanilla *Clingo*

- Emulating *Clingo* in *Clingo* 4

```
#script (python)
def main(prg):
    parts = []
    parts.append(("base", []))
    prg.ground(parts)
    prg.solve()
#end.
```

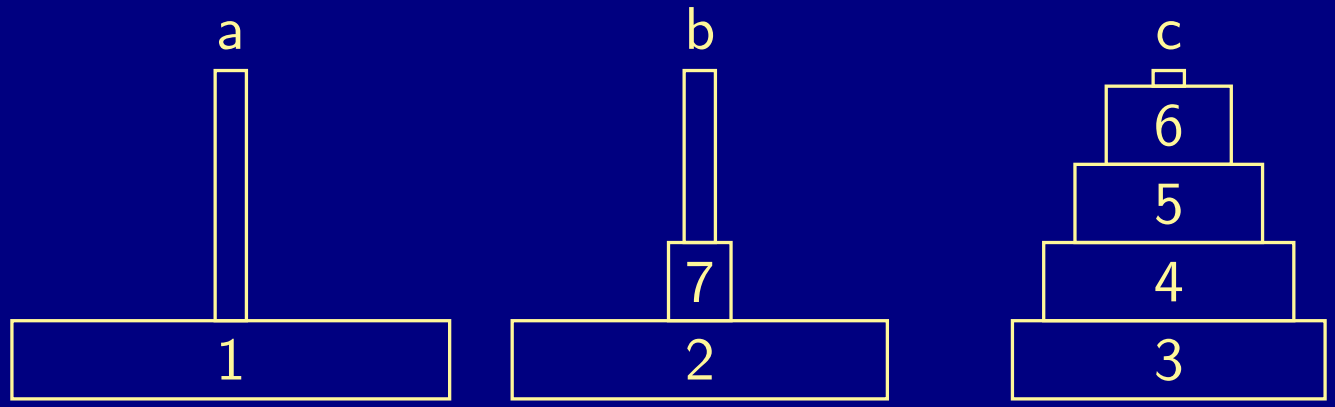
Outline

Potassco

Towers of Hanoi Instance

- Emulating *iClingo* in *Clingo* 4
  - Incremental grounding
  - Incremental solving

Towers of Hanoi Instance




```
peg(a;b;c).
init_on(1,a).
init_on((2;7),b).
init_on((3;4;5;6),c).

disk(1..7).

goal_on((3;4),a).
goal_on((1;2;5;6;7),c).
```

Potassco

Towers of Hanoi Encoding (base)

```
#program base.

on(D,P,0) :- init_on(D,P).
```

Towers of Hanoi Encoding (cumulative)

```
#program cumulative(t).

1 { move(D,P,t) : disk(D), peg(P) } 1.

moved(D,t) :- move(D,_,t).
blocked(D,P,t) :- on(D+1,P,t-1), disk(D).
blocked(D,P,t) :- blocked(D+1,P,t), disk(D).
:- move(D,P,t), blocked(D-1,P,t).
:- moved(D,t), on(D,P,t-1), blocked(D,P,t).

on(D,P,t) :- on(D,P,t-1), not moved(D,t).
on(D,P,t) :- move(D,P,t).
:- not 1 { on(D,P,t) : peg(P) } 1, disk(D).
```

Potassco

Towers of Hanoi Encoding (volatile)

```
#program volatile(t).

#external query(t).

:- goal_on(D,P), not on(D,P,t), query(t).
```

Incremental Solving (embedded)

```
#script (python)

from gringo import SolveResult, Fun

def main(prg):
    ret, parts, step = SolveResult.UNSAT, [], 1
    parts.append(("base", []))
    while ret == SolveResult.UNSAT:
        parts.append(("cumulative", [step]))
        parts.append(("volatile", [step]))
        prg.ground(parts)
        prg.release_external(Fun("query", [step-1]))
        prg.assign_external(Fun("query", [step]), True)
        ret, parts, step = prg.solve(), [], step+1

#end.
```

Potassco

Incremental Solving (library)

```
from sys import stdout
from gringo import SolveResult, Fun, Control

prg = Control()
prg.load("toh.lp")

ret, parts, step = SolveResult.UNSAT, [], 1
parts.append(("base", []))
while ret == SolveResult.UNSAT:
    parts.append(("cumulative", [step]))
    parts.append(("volatile", [step]))
    prg.ground(parts)
    prg.release_external(Fun("query", [step-1]))
    prg.assign_external(Fun("query", [step]), True)
    f = lambda m: stdout.write(str(m))
    ret, parts, step = prg.solve(on_model=f), [], step+1
```

Potassco

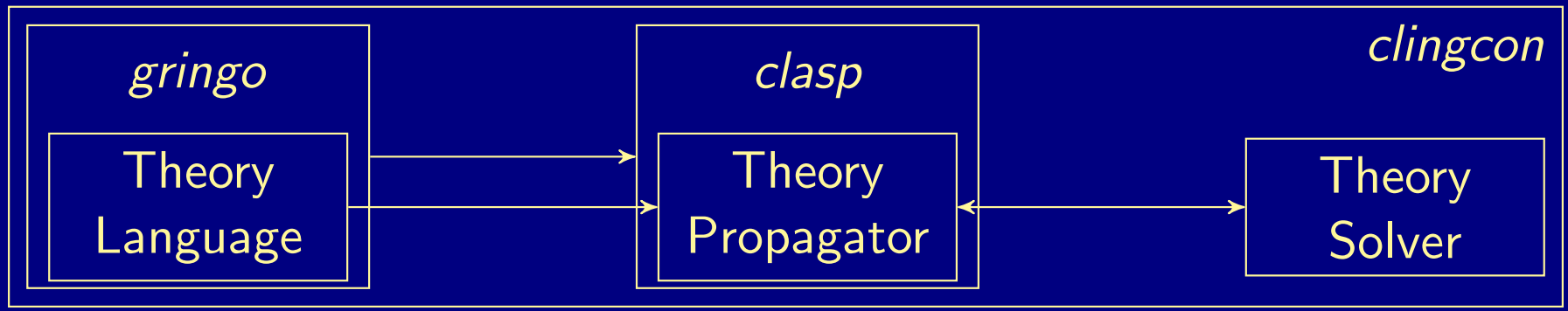
Outline

Potassco

clingcon

- Hybrid grounding and solving
- Solving in hybrid domains, like Bio-Informatics

- Basic architecture of *clingcon*:



Pouring Water into Buckets on a Scale

```
time(0..t).                 $domain(0..500).
bucket(a).                  volume(a,0) $== 0.
bucket(b).                  volume(b,0) $== 100.

1 { pour(B,T) : bucket(B) } 1 :- time(T), T < t.

          1 $<= amount(B,T) :- pour(B,T), T < t.
amount(B,T) $<= 30          :- pour(B,T), T < t.
amount(B,T) $==  0          :- not pour(B,T), bucket(B), time(T), T < t.

volume(B,T+1) $== volume(B,T) $+ amount(B,T) :- bucket(B), time(T), T < t.

down(B,T) :- volume(C,T) $< volume(B,T), bucket(B;C), time(T).
  up(B,T) :- not down(B,T), bucket(B), time(T).

 :- up(a,t).
```

Potassco

Pouring Water into Buckets on a Scale

```
time(0..t).                $domain(0..500).
bucket(a).                 volume(a,0) $== 0.
bucket(b).                 volume(b,0) $== 100.

1 { pour(B,T) : bucket(B) } 1 :- time(T), T < t.

          1 $<= amount(B,T) :- pour(B,T), T < t.
amount(B,T) $<= 30          :- pour(B,T), T < t.
amount(B,T) $==  0          :- not pour(B,T), bucket(B), time(T), T < t.

volume(B,T+1) $== volume(B,T) $+ amount(B,T) :- bucket(B), time(T), T < t.

down(B,T) :- volume(C,T) $< volume(B,T), bucket(B;C), time(T).
  up(B,T) :- not down(B,T), bucket(B), time(T).

 :- up(a,t).
```

Pouring Water into Buckets on a Scale

```
time(0..t).                $domain(0..500).
bucket(a).                 volume(a,0) $== 0.
bucket(b).                 volume(b,0) $== 100.

1 { pour(B,T) : bucket(B) } 1 :- time(T), T < t.

          1 $<= amount(B,T) :- pour(B,T), T < t.
amount(B,T) $<= 30          :- pour(B,T), T < t.
amount(B,T) $==  0          :- not pour(B,T), bucket(B), time(T), T < t.

volume(B,T+1) $== volume(B,T) $+ amount(B,T) :- bucket(B), time(T), T < t.

down(B,T) :- volume(C,T) $< volume(B,T), bucket(B;C), time(T).
  up(B,T) :- not down(B,T), bucket(B), time(T).

 :- up(a,t).
```

Potassco

Pouring Water into Buckets on a Scale

```
time(0..t).                $domain(0..500).
bucket(a).                 volume(a,0) $== 0.
bucket(b).                 volume(b,0) $== 100.

1 { pour(B,T) : bucket(B) } 1 :- time(T), T < t.

          1 $<= amount(B,T) :- pour(B,T), T < t.
amount(B,T) $<= 30          :- pour(B,T), T < t.
amount(B,T) $==  0          :- not pour(B,T), bucket(B), time(T), T < t.

volume(B,T+1) $== volume(B,T) $+ amount(B,T) :- bucket(B), time(T), T < t.

down(B,T) :- volume(C,T) $< volume(B,T), bucket(B;C), time(T).
  up(B,T) :- not down(B,T), bucket(B), time(T).

 :- up(a,t).
```

Pouring Water into Buckets on a Scale

```
time(0..t).                 $domain(0..500).
bucket(a).                  volume(a,0) $== 0.
bucket(b).                  volume(b,0) $== 100.

1 { pour(B,T) : bucket(B) } 1 :- time(T), T < t.

 :- pour(B,T), T < t, not (1 $<= amount(B,T)).
amount(B,T) $<= 30            :- pour(B,T), T < t.
amount(B,T) $==  0            :- not pour(B,T), bucket(B), time(T), T < t.

volume(B,T+1) $== volume(B,T) $+ amount(B,T) :- bucket(B), time(T), T < t.

down(B,T) :- volume(C,T) $< volume(B,T), bucket(B;C), time(T).
  up(B,T) :- not down(B,T), bucket(B), time(T).

 :- up(a,t).
```

Potassco

Pouring Water into Buckets on a Scale

```
time(0..t).                 $domain(0..500).
bucket(a).                  volume(a,0) $== 0.
bucket(b).                  volume(b,0) $== 100.

1 { pour(B,T) : bucket(B) } 1 :- time(T), T < t.

 :- pour(B,T), T < t, 1 $> amount(B,T).
amount(B,T) $<= 30            :- pour(B,T), T < t.
amount(B,T) $==  0            :- not pour(B,T), bucket(B), time(T), T < t.

volume(B,T+1) $== volume(B,T) $+ amount(B,T) :- bucket(B), time(T), T < t.

down(B,T) :- volume(C,T) $< volume(B,T), bucket(B;C), time(T).
  up(B,T) :- not down(B,T), bucket(B), time(T).

 :- up(a,t).
```

Potassco

Pouring Water into Buckets on a Scale

```
time(0..t).                $domain(0..500).
bucket(a).                 volume(a,0) $== 0.
bucket(b).                 volume(b,0) $== 100.

1 { pour(B,T) : bucket(B) } 1 :- time(T), T < t.

 :- pour(B,T), T < t, 1 $> amount(B,T).
 :- pour(B,T), T < t, amount(B,T) $> 30.
amount(B,T) $==  0          :- not pour(B,T), bucket(B), time(T), T < t.

volume(B,T+1) $== volume(B,T) $+ amount(B,T) :- bucket(B), time(T), T < t.

down(B,T) :- volume(C,T) $< volume(B,T), bucket(B;C), time(T).
  up(B,T) :- not down(B,T), bucket(B), time(T).

 :- up(a,t).
```

Potassco

Pouring Water into Buckets on a Scale

```
time(0..t).                 $domain(0..500).
bucket(a).                  volume(a,0) $== 0.
bucket(b).                  volume(b,0) $== 100.

1 { pour(B,T) : bucket(B) } 1 :- time(T), T < t.

 :- pour(B,T), T < t, 1 $> amount(B,T).
 :- pour(B,T), T < t, amount(B,T) $> 30.
 :- not pour(B,T), bucket(B), time(T), T < t, amount(B,T) $!=  0.

volume(B,T+1) $== volume(B,T) $+ amount(B,T) :- bucket(B), time(T), T < t.

down(B,T) :- volume(C,T) $< volume(B,T), bucket(B;C), time(T).
  up(B,T) :- not down(B,T), bucket(B), time(T).

 :- up(a,t).
```

Pouring Water into Buckets on a Scale

```
time(0..t).                 $domain(0..500).
bucket(a).                  volume(a,0) $== 0.
bucket(b).                  volume(b,0) $== 100.

1 { pour(B,T) : bucket(B) } 1 :- time(T), T < t.

 :- pour(B,T), T < t, 1 $> amount(B,T).
 :- pour(B,T), T < t, amount(B,T) $> 30.
 :- not pour(B,T), bucket(B), time(T), T < t, amount(B,T) $!=  0.

 :- bucket(B), time(T), T < t, volume(B,T+1) $!= volume(B,T)$+amount(B,T).

down(B,T) :- volume(C,T) $< volume(B,T), bucket(B;C), time(T).
  up(B,T) :- not down(B,T), bucket(B), time(T).

 :- up(a,t).
```

Potassco

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp --text
```

```
time(0). ... time(4).
bucket(a).
bucket(b).

1 { pour(b,0), pour(a,0) } 1.

 :- pour(a,0), 1 $> amount(a,0).
 :- pour(b,0), 1 $> amount(b,0).

 :- pour(a,0), amount(a,0) $> 30.
 :- pour(b,0), amount(b,0) $> 30.

 :- not pour(a,0), amount(a,0) $!= 0.
 :- not pour(b,0), amount(b,0) $!= 0.

 :- volume(a,1) $!= (volume(a,0) $+ amount(a,0)).
 :- volume(b,1) $!= (volume(b,0) $+ amount(b,0)).

down(a,0) :- volume(a,0) $< volume(a,0).
down(a,0) :- volume(b,0) $< volume(a,0).
down(b,0) :- volume(a,0) $< volume(b,0).
down(b,0) :- volume(b,0) $< volume(b,0).

up(a,0) :- not down(a,0).
up(b,0) :- not down(b,0).

 :- up(a,4).
```

```
$domain(0..500).
 :- volume(a,0) $!= 0.
 :- volume(b,0) $!= 100.

... 1 { pour(b,3), pour(a,3) } 1.

...  :- pour(a,3), 1 $> amount(a,3).
...  :- pour(b,3), 1 $> amount(b,3).

...  :- pour(a,3), amount(a,3) $> 30.
...  :- pour(b,3), amount(b,3) $> 30.

...  :- not pour(a,3), amount(a,3) $!= 0.
...  :- not pour(b,3), amount(b,3) $!= 0.

...  :- volume(a,4) $!= (volume(a,3) $+ amount(a,3)).
...  :- volume(b,4) $!= (volume(b,3) $+ amount(b,3)).

... down(a,4) :- volume(a,4) $< volume(a,4).
... down(a,4) :- volume(b,4) $< volume(a,4).
... down(b,4) :- volume(a,4) $< volume(b,4).
... down(b,4) :- volume(b,4) $< volume(b,4).

... up(a,4) :- not down(a,4).
... up(b,4) :- not down(b,4).
```

Potassco

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp --text
```

```
time(0). ... time(4).
bucket(a).
bucket(b).

1 { pour(b,0), pour(a,0) } 1.

 :- pour(a,0), 1 $> amount(a,0).
 :- pour(b,0), 1 $> amount(b,0).

 :- pour(a,0), amount(a,0) $> 30.
 :- pour(b,0), amount(b,0) $> 30.

 :- not pour(a,0), amount(a,0) $!= 0.
 :- not pour(b,0), amount(b,0) $!= 0.

 :- volume(a,1) $!= (volume(a,0) $+ amount(a,0)).
 :- volume(b,1) $!= (volume(b,0) $+ amount(b,0)).

down(a,0) :- volume(a,0) $< volume(a,0).
down(a,0) :- volume(b,0) $< volume(a,0).
down(b,0) :- volume(a,0) $< volume(b,0).
down(b,0) :- volume(b,0) $< volume(b,0).

up(a,0) :- not down(a,0).
up(b,0) :- not down(b,0).

 :- up(a,4).
```

```
$domain(0..500).
 :- volume(a,0) $!= 0.
 :- volume(b,0) $!= 100.

... 1 { pour(b,3), pour(a,3) } 1.

...  :- pour(a,3), 1 $> amount(a,3).
...  :- pour(b,3), 1 $> amount(b,3).

...  :- pour(a,3), amount(a,3) $> 30.
...  :- pour(b,3), amount(b,3) $> 30.

...  :- not pour(a,3), amount(a,3) $!= 0.
...  :- not pour(b,3), amount(b,3) $!= 0.

...  :- volume(a,4) $!= (volume(a,3) $+ amount(a,3)).
...  :- volume(b,4) $!= (volume(b,3) $+ amount(b,3)).

... down(a,4) :- volume(a,4) $< volume(a,4).
... down(a,4) :- volume(b,4) $< volume(a,4).
... down(b,4) :- volume(a,4) $< volume(b,4).
... down(b,4) :- volume(b,4) $< volume(b,4).

... up(a,4) :- not down(a,4).
... up(b,4) :- not down(b,4).
```

Potassco

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp --text
```

```
time(0). ... time(4).
bucket(a).
bucket(b).

1 { pour(b,0), pour(a,0) } 1.

 :- pour(a,0), 1 $> amount(a,0).
 :- pour(b,0), 1 $> amount(b,0).

 :- pour(a,0), amount(a,0) $> 30.
 :- pour(b,0), amount(b,0) $> 30.

 :- not pour(a,0), amount(a,0) $!= 0.
 :- not pour(b,0), amount(b,0) $!= 0.

 :- volume(a,1) $!= (volume(a,0) $+ amount(a,0)).
 :- volume(b,1) $!= (volume(b,0) $+ amount(b,0)).

down(a,0) :- volume(a,0) $< volume(a,0).
down(a,0) :- volume(b,0) $< volume(a,0).
down(b,0) :- volume(a,0) $< volume(b,0).
down(b,0) :- volume(b,0) $< volume(b,0).

up(a,0) :- not down(a,0).
up(b,0) :- not down(b,0).

 :- up(a,4).
```

```
$domain(0..500).
 :- volume(a,0) $!= 0.
 :- volume(b,0) $!= 100.

... 1 { pour(b,3), pour(a,3) } 1.

...  :- pour(a,3), 1 $> amount(a,3).
...  :- pour(b,3), 1 $> amount(b,3).

...  :- pour(a,3), amount(a,3) $> 30.
...  :- pour(b,3), amount(b,3) $> 30.

...  :- not pour(a,3), amount(a,3) $!= 0.
...  :- not pour(b,3), amount(b,3) $!= 0.

...  :- volume(a,4) $!= (volume(a,3) $+ amount(a,3)).
...  :- volume(b,4) $!= (volume(b,3) $+ amount(b,3)).

... down(a,4) :- volume(a,4) $< volume(a,4).
... down(a,4) :- volume(b,4) $< volume(a,4).
... down(b,4) :- volume(a,4) $< volume(b,4).
... down(b,4) :- volume(b,4) $< volume(b,4).

... up(a,4) :- not down(a,4).
... up(b,4) :- not down(b,4).
```

Potassco

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp --text
```

```
time(0). ... time(4).
bucket(a).
bucket(b).

1 { pour(b,0), pour(a,0) } 1.

 :- pour(a,0), 1 $> amount(a,0).
 :- pour(b,0), 1 $> amount(b,0).

 :- pour(a,0), amount(a,0) $> 30.
 :- pour(b,0), amount(b,0) $> 30.

 :- not pour(a,0), amount(a,0) $!= 0.
 :- not pour(b,0), amount(b,0) $!= 0.

 :- volume(a,1) $!= (volume(a,0) $+ amount(a,0)).
 :- volume(b,1) $!= (volume(b,0) $+ amount(b,0)).

down(a,0) :- volume(a,0) $< volume(a,0).
down(a,0) :- volume(b,0) $< volume(a,0).
down(b,0) :- volume(a,0) $< volume(b,0).
down(b,0) :- volume(b,0) $< volume(b,0).

up(a,0) :- not down(a,0).
up(b,0) :- not down(b,0).

 :- up(a,4).
```

```
$domain(0..500).
 :- volume(a,0) $!= 0.
 :- volume(b,0) $!= 100.

... 1 { pour(b,3), pour(a,3) } 1.

...  :- pour(a,3), 1 $> amount(a,3).
...  :- pour(b,3), 1 $> amount(b,3).

...  :- pour(a,3), amount(a,3) $> 30.
...  :- pour(b,3), amount(b,3) $> 30.

...  :- not pour(a,3), amount(a,3) $!= 0.
...  :- not pour(b,3), amount(b,3) $!= 0.

...  :- volume(a,4) $!= (volume(a,3) $+ amount(a,3)).
...  :- volume(b,4) $!= (volume(b,3) $+ amount(b,3)).

... down(a,4) :- volume(a,4) $< volume(a,4).
... down(a,4) :- volume(b,4) $< volume(a,4).
... down(b,4) :- volume(a,4) $< volume(b,4).
... down(b,4) :- volume(b,4) $< volume(b,4).

... up(a,4) :- not down(a,4).
... up(b,4) :- not down(b,4).
```

Potassco

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp --text
```

```
time(0). ... time(4).
bucket(a).
bucket(b).

1 { pour(b,0), pour(a,0) } 1.

 :- pour(a,0), 1 $> amount(a,0).
 :- pour(b,0), 1 $> amount(b,0).

 :- pour(a,0), amount(a,0) $> 30.
 :- pour(b,0), amount(b,0) $> 30.

 :- not pour(a,0), amount(a,0) $!= 0.
 :- not pour(b,0), amount(b,0) $!= 0.

 :- volume(a,1) $!= (volume(a,0) $+ amount(a,0)).
 :- volume(b,1) $!= (volume(b,0) $+ amount(b,0)).

down(a,0) :- volume(a,0) $< volume(a,0).
down(a,0) :- volume(b,0) $< volume(a,0).
down(b,0) :- volume(a,0) $< volume(b,0).
down(b,0) :- volume(b,0) $< volume(b,0).

up(a,0) :- not down(a,0).
up(b,0) :- not down(b,0).

 :- up(a,4).
```

```
$domain(0..500).
 :- volume(a,0) $!= 0.
 :- volume(b,0) $!= 100.

... 1 { pour(b,3), pour(a,3) } 1.

...  :- pour(a,3), 1 $> amount(a,3).
...  :- pour(b,3), 1 $> amount(b,3).

...  :- pour(a,3), amount(a,3) $> 30.
...  :- pour(b,3), amount(b,3) $> 30.

...  :- not pour(a,3), amount(a,3) $!= 0.
...  :- not pour(b,3), amount(b,3) $!= 0.

...  :- volume(a,4) $!= (volume(a,3) $+ amount(a,3)).
...  :- volume(b,4) $!= (volume(b,3) $+ amount(b,3)).

... down(a,4) :- volume(a,4) $< volume(a,4).
... down(a,4) :- volume(b,4) $< volume(a,4).
... down(b,4) :- volume(a,4) $< volume(b,4).
... down(b,4) :- volume(b,4) $< volume(b,4).

... up(a,4) :- not down(a,4).
... up(b,4) :- not down(b,4).
```

Potassco

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp 0

Answer: 1
pour(a,0)   pour(a,1)   pour(a,2)   pour(a,3)

amount(a,0)=[11..30]   amount(b,0)=0          1 $> amount(b,0)   amount(a,0) $!= 0
amount(a,1)=[11..30]   amount(b,1)=0          1 $> amount(b,1)   amount(a,1) $!= 0
amount(a,2)=[11..30]   amount(b,2)=0          1 $> amount(b,2)   amount(a,2) $!= 0
amount(a,3)=[11..30]   amount(b,3)=0          1 $> amount(b,3)   amount(a,3) $!= 0

volume(a,0)=0          volume(b,0)=100      volume(a,0) $< volume(b,0)
volume(a,1)=[11..30]   volume(b,1)=100      volume(a,1) $< volume(b,1)
volume(a,2)=[41..60]   volume(b,2)=100      volume(a,2) $< volume(b,2)
volume(a,3)=[71..90]   volume(b,3)=100      volume(a,3) $< volume(b,3)
volume(a,4)=[101..120] volume(b,4)=100      volume(b,4) $< volume(a,4)

SATISFIABLE

Models       : 1
Time         : 0.000
```

Potassco

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp 0

Answer: 1
pour(a,0)   pour(a,1)   pour(a,2)   pour(a,3)

amount(a,0)=[11..30]   amount(b,0)=0          1 $> amount(b,0)   amount(a,0) $!= 0
amount(a,1)=[11..30]   amount(b,1)=0          1 $> amount(b,1)   amount(a,1) $!= 0
amount(a,2)=[11..30]   amount(b,2)=0          1 $> amount(b,2)   amount(a,2) $!= 0
amount(a,3)=[11..30]   amount(b,3)=0          1 $> amount(b,3)   amount(a,3) $!= 0

volume(a,0)=0          volume(b,0)=100      volume(a,0) $< volume(b,0)
volume(a,1)=[11..30]   volume(b,1)=100      volume(a,1) $< volume(b,1)
volume(a,2)=[41..60]   volume(b,2)=100      volume(a,2) $< volume(b,2)
volume(a,3)=[71..90]   volume(b,3)=100      volume(a,3) $< volume(b,3)
volume(a,4)=[101..120] volume(b,4)=100      volume(b,4) $< volume(a,4)

SATISFIABLE

Models       : 1
Time         : 0.000
```

Potassco

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp 0

Answer: 1
pour(a,0)   pour(a,1)   pour(a,2)   pour(a,3)

amount(a,0)=[11..30]   amount(b,0)=0          1 $> amount(b,0)   amount(a,0) $!= 0
amount(a,1)=[11..30]   amount(b,1)=0          1 $> amount(b,1)   amount(a,1) $!= 0
amount(a,2)=[11..30]   amount(b,2)=0          1 $> amount(b,2)   amount(a,2) $!= 0
amount(a,3)=[11..30]   amount(b,3)=0          1 $> amount(b,3)   amount(a,3) $!= 0

volume(a,0)=0          volume(b,0)=100      volume(a,0) $< volume(b,0)
volume(a,1)=[11..30]   volume(b,1)=100      volume(a,1) $< volume(b,1)
volume(a,2)=[41..60]   volume(b,2)=100      volume(a,2) $< volume(b,2)
volume(a,3)=[71..90]   volume(b,3)=100      volume(a,3) $< volume(b,3)
volume(a,4)=[101..120] volume(b,4)=100      volume(b,4) $< volume(a,4)

SATISFIABLE

Models       : 1
Time         : 0.000
```

Potassco

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp 0

Answer: 1
pour(a,0)   pour(a,1)   pour(a,2)   pour(a,3)

amount(a,0)=[11..30]   amount(b,0)=0          1 $> amount(b,0)   amount(a,0) $!= 0
amount(a,1)=[11..30]   amount(b,1)=0          1 $> amount(b,1)   amount(a,1) $!= 0
amount(a,2)=[11..30]   amount(b,2)=0          1 $> amount(b,2)   amount(a,2) $!= 0
amount(a,3)=[11..30]   amount(b,3)=0          1 $> amount(b,3)   amount(a,3) $!= 0

volume(a,0)=0          volume(b,0)=100      volume(a,0) $< volume(b,0)
volume(a,1)=[11..30]   volume(b,1)=100      volume(a,1) $< volume(b,1)
volume(a,2)=[41..60]   volume(b,2)=100      volume(a,2) $< volume(b,2)
volume(a,3)=[71..90]   volume(b,3)=100      volume(a,3) $< volume(b,3)
volume(a,4)=[101..120] volume(b,4)=100      volume(b,4) $< volume(a,4)

SATISFIABLE

Models       : 1
Time         : 0.000
```

Potassco

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp 0

Answer: 1
pour(a,0)   pour(a,1)   pour(a,2)   pour(a,3)

amount(a,0)=[11..30]   amount(b,0)=0          1 $> amount(b,0)   amount(a,0) $!= 0
amount(a,1)=[11..30]   amount(b,1)=0          1 $> amount(b,1)   amount(a,1) $!= 0
amount(a,2)=[11..30]   amount(b,2)=0          1 $> amount(b,2)   amount(a,2) $!= 0
amount(a,3)=[11..30]   amount(b,3)=0          1 $> amount(b,3)   amount(a,3) $!= 0

volume(a,0)=0          volume(b,0)=100     volume(a,0) $< volume(b,0)
volume(a,1)=[11..30]   volume(b,1)=100     volume(a,1) $< volume(b,1)
volume(a,2)=[41..60]   volume(b,2)=100     volume(a,2) $< volume(b,2)
volume(a,3)=[71..90]   volume(b,3)=100     volume(a,3) $< volume(b,3)
volume(a,4)=[101..120] volume(b,4)=100     volume(b,4) $< volume(a,4)

SATISFIABLE

Models       : 1
Time         : 0.000
```

Boolean variables

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp 0

Answer: 1
pour(a,0)   pour(a,1)   pour(a,2)   pour(a,3)

amount(a,0)=[11..30]   amount(b,0)=0          1 $> amount(b,0)   amount(a,0) $!= 0
amount(a,1)=[11..30]   amount(b,1)=0          1 $> amount(b,1)   amount(a,1) $!= 0
amount(a,2)=[11..30]   amount(b,2)=0          1 $> amount(b,2)   amount(a,2) $!= 0
amount(a,3)=[11..30]   amount(b,3)=0          1 $> amount(b,3)   amount(a,3) $!= 0

volume(a,0)=0          volume(b,0)=100     volume(a,0) $< volume(b,0)
volume(a,1)=[11..30]   volume(b,1)=100     volume(a,1) $< volume(b,1)
volume(a,2)=[41..60]   volume(b,2)=100     volume(a,2) $< volume(b,2)
volume(a,3)=[71..90]   volume(b,3)=100     volume(a,3) $< volume(b,3)
volume(a,4)=[101..120] volume(b,4)=100     volume(b,4) $< volume(a,4)

SATISFIABLE

Models       : 1
Time         : 0.000
```

Non-Boolean variables

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp --csp-num-as=1

Answer: 1
pour(a,0)   pour(a,1)   pour(a,2)   pour(a,3)

amount(a,0)=11          amount(b,0)=0          1 $> amount(b,0)   amount(a,0) $!= 0
amount(a,1)=30          amount(b,1)=0          1 $> amount(b,1)   amount(a,1) $!= 0
amount(a,2)=30          amount(b,2)=0          1 $> amount(b,2)   amount(a,2) $!= 0
amount(a,3)=30          amount(b,3)=0          1 $> amount(b,3)   amount(a,3) $!= 0

volume(a,0)=0           volume(b,0)=100        volume(a,0) $< volume(b,0)
volume(a,1)=11          volume(b,1)=100        volume(a,1) $< volume(b,1)
volume(a,2)=41          volume(b,2)=100        volume(a,2) $< volume(b,2)
volume(a,3)=71          volume(b,3)=100        volume(a,3) $< volume(b,3)
volume(a,4)=101         volume(b,4)=100        volume(b,4) $< volume(a,4)

SATISFIABLE

Models       : 1+
Time         : 0.000
```

Potassco

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp --csp-num-as=1

Answer: 1
pour(a,0)   pour(a,1)   pour(a,2)   pour(a,3)

amount(a,0)=11          amount(b,0)=0          1 $> amount(b,0)   amount(a,0) $!= 0
amount(a,1)=30          amount(b,1)=0          1 $> amount(b,1)   amount(a,1) $!= 0
amount(a,2)=30          amount(b,2)=0          1 $> amount(b,2)   amount(a,2) $!= 0
amount(a,3)=30          amount(b,3)=0          1 $> amount(b,3)   amount(a,3) $!= 0

volume(a,0)=0           volume(b,0)=100        volume(a,0) $< volume(b,0)
volume(a,1)=11          volume(b,1)=100        volume(a,1) $< volume(b,1)
volume(a,2)=41          volume(b,2)=100        volume(a,2) $< volume(b,2)
volume(a,3)=71          volume(b,3)=100        volume(a,3) $< volume(b,3)
volume(a,4)=101         volume(b,4)=100        volume(b,4) $< volume(a,4)

SATISFIABLE

Models       : 1+
Time         : 0.000
```

Potassco

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp --csp-num-as=1

Answer: 1
pour(a,0)   pour(a,1)   pour(a,2)   pour(a,3)

amount(a,0)=11          amount(b,0)=0          1 $> amount(b,0)   amount(a,0) $!= 0
amount(a,1)=30          amount(b,1)=0          1 $> amount(b,1)   amount(a,1) $!= 0
amount(a,2)=30          amount(b,2)=0          1 $> amount(b,2)   amount(a,2) $!= 0
amount(a,3)=30          amount(b,3)=0          1 $> amount(b,3)   amount(a,3) $!= 0

volume(a,0)=0           volume(b,0)=100        volume(a,0) $< volume(b,0)
volume(a,1)=11          volume(b,1)=100        volume(a,1) $< volume(b,1)
volume(a,2)=41          volume(b,2)=100        volume(a,2) $< volume(b,2)
volume(a,3)=71          volume(b,3)=100        volume(a,3) $< volume(b,3)
volume(a,4)=101         volume(b,4)=100        volume(b,4) $< volume(a,4)

SATISFIABLE

Models       : 1+
Time         : 0.000
```

Potassco

Pouring Water into Buckets on a Scale

```
$ clingcon --const t=4 balance.lp --csp-num-as=1

Answer: 1
pour(a,0)   pour(a,1)   pour(a,2)   pour(a,3)

amount(a,0)=11          amount(b,0)=0          1 $> amount(b,0)   amount(a,0) $!= 0
amount(a,1)=30          amount(b,1)=0          1 $> amount(b,1)   amount(a,1) $!= 0
amount(a,2)=30          amount(b,2)=0          1 $> amount(b,2)   amount(a,2) $!= 0
amount(a,3)=30          amount(b,3)=0          1 $> amount(b,3)   amount(a,3) $!= 0

volume(a,0)=0           volume(b,0)=100        volume(a,0) $< volume(b,0)
volume(a,1)=11          volume(b,1)=100        volume(a,1) $< volume(b,1)
volume(a,2)=41          volume(b,2)=100        volume(a,2) $< volume(b,2)
volume(a,3)=71          volume(b,3)=100        volume(a,3) $< volume(b,3)
volume(a,4)=101         volume(b,4)=100        volume(b,4) $< volume(a,4)

SATISFIABLE

Models       : 1+
Time         : 0.000
```

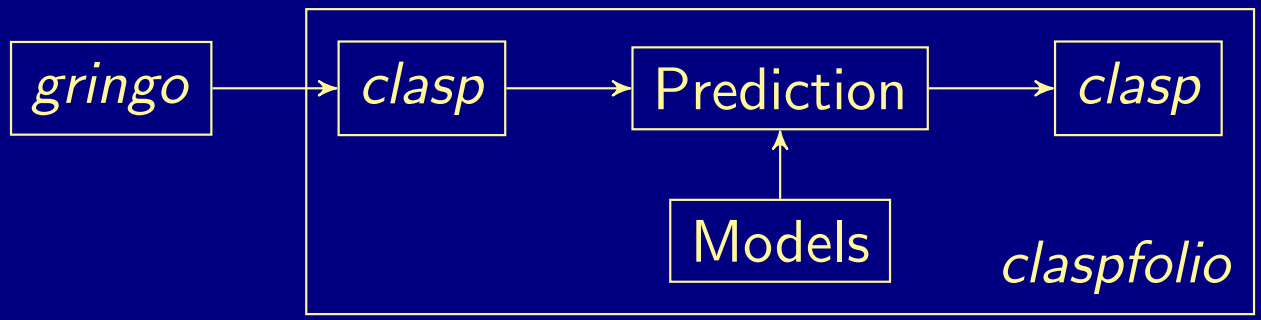
Potassco

Outline

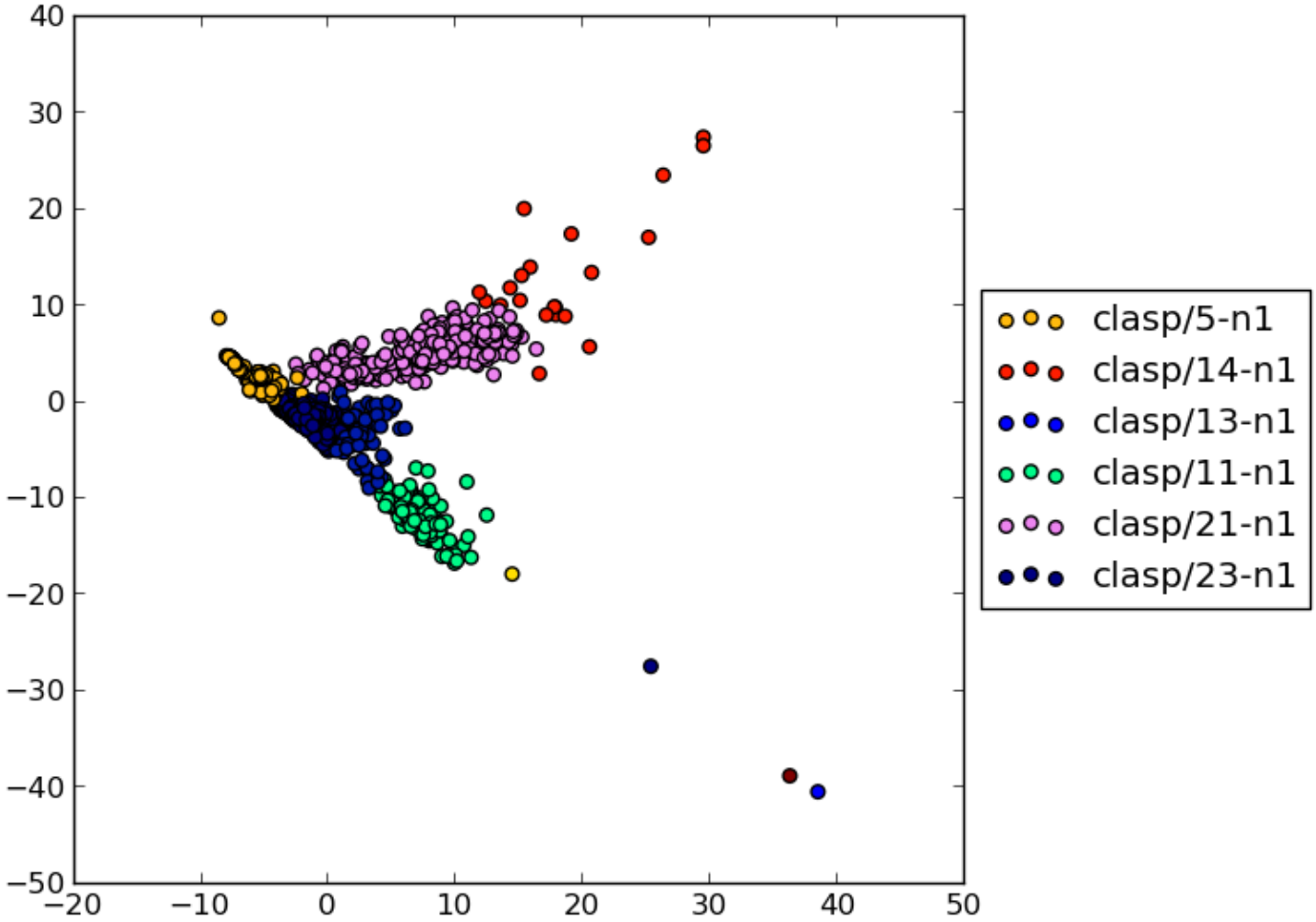
Potassco

claspfolio

- Automatic selection of some *clasp* configuration among several predefined ones via (learned) classifiers
- Basic architecture of *claspfolio*:



Instance Feature Clusters (after PCA)



Potassco

Solving with *clasp* (as usual)

```
$ clasp queens500 --quiet

clasp version 2.0.2
Reading from queens500
Solving...
SATISFIABLE

Models       : 1+
Time         : 11.445s (Solving: 10.58s 1st Model: 10.55s Unsat: 0.00s)
CPU Time     : 11.410s
```

Potassco

Solving with *clasp* (as usual)

```
$ clasp queens500 --quiet

clasp version 2.0.2
Reading from queens500
Solving...
SATISFIABLE

Models       : 1+
Time         : 11.445s (Solving: 10.58s 1st Model: 10.55s Unsat: 0.00s)
CPU Time     : 11.410s
```

Potassco

Solving with *claspfolio*

```
$ claspfolio queens500 --quiet

PRESOLVING
Reading from queens500
Solving...
claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models       : 1+
Time         : 4.785s (Solving: 3.96s 1st Model: 3.92s Unsat: 0.00s)
CPU Time     : 4.780s
```

Potassco

Solving with *claspfolio*

```
$ claspfolio queens500 --quiet

PRESOLVING
Reading from queens500
Solving...
claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models       : 1+
Time         : 4.785s (Solving: 3.96s 1st Model: 3.92s Unsat: 0.00s)
CPU Time     : 4.780s
```

Potassco

Solving with *claspfolio*

```
$ claspfolio queens500 --quiet

PRESOLVING
Reading from queens500
Solving...
claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models       : 1+
Time         : 4.785s (Solving: 3.96s 1st Model: 3.92s Unsat: 0.00s)
CPU Time     : 4.780s
```

Potassco

Solving with *claspfolio*

```
$ claspfolio queens500 --quiet

PRESOLVING
Reading from queens500
Solving...
claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models       : 1+
Time         : 4.785s (Solving: 3.96s 1st Model: 3.92s Unsat: 0.00s)
CPU Time     : 4.780s
```

Potassco

Feature-extraction with *claspfolio*

```
$ claspfolio --features queens500

PRESOLVING
Reading from queens500
Solving...
UNKNOWN
Features    : 84998,3994,0,250000,1.020,62.594,63.844,21.281,84998, \
 3994,100,250000,1.020,62.594,63.844,21.281,84998,3994,250,250000,  \
 1.020,62.594,63.844,21.281,84998,3994,475,250000,1.020,62.594,     \
 63.844,21.281,757989,757989,0,510983,506992,3990,1,0,127.066,9983, \
 1023958,502993,1994,518971,1,0,0,254994,0,3990,0.100,0.000,99.900, \
 0,270303,812,4,0,812,2223,2223,262,262,2.738,2.738,0.000,812,812,  \
 2270.982,0,0.000

$ claspfolio --list-features

maxLearnt,Constraints,LearntConstraints,FreeVars,Vars/FreeVars, ...
```

Potassco

Feature-extraction with *claspfolio*

```
$ claspfolio --features queens500

PRESOLVING
Reading from queens500
Solving...
UNKNOWN
Features    : 84998,3994,0,250000,1.020,62.594,63.844,21.281,84998, \
 3994,100,250000,1.020,62.594,63.844,21.281,84998,3994,250,250000,  \
 1.020,62.594,63.844,21.281,84998,3994,475,250000,1.020,62.594,     \
 63.844,21.281,757989,757989,0,510983,506992,3990,1,0,127.066,9983, \
 1023958,502993,1994,518971,1,0,0,254994,0,3990,0.100,0.000,99.900, \
 0,270303,812,4,0,812,2223,2223,262,262,2.738,2.738,0.000,812,812,  \
 2270.982,0,0.000

$ claspfolio --list-features

maxLearnt,Constraints,LearntConstraints,FreeVars,Vars/FreeVars, ...
```

Potassco

Feature-extraction with *claspfolio*

```
$ claspfolio --features queens500

PRESOLVING
Reading from queens500
Solving...
UNKNOWN
Features    : 84998,3994,0,250000,1.020,62.594,63.844,21.281,84998, \
 3994,100,250000,1.020,62.594,63.844,21.281,84998,3994,250,250000,  \
 1.020,62.594,63.844,21.281,84998,3994,475,250000,1.020,62.594,     \
 63.844,21.281,757989,757989,0,510983,506992,3990,1,0,127.066,9983, \
 1023958,502993,1994,518971,1,0,0,254994,0,3990,0.100,0.000,99.900, \
 0,270303,812,4,0,812,2223,2223,262,262,2.738,2.738,0.000,812,812,  \
 2270.982,0,0.000

$ claspfolio --list-features

maxLearnt,Constraints,LearntConstraints,FreeVars,Vars/FreeVars, ...
```

Potassco

Prediction with *claspfolio*

```
$ claspfolio queens500 --decisionvalues

PRESOLVING
Reading from queens500
Solving...

Portfolio Decision Values:
[1] : 3.437538   [10] : 3.639444   [19] : 3.726391
[2] : 3.501728   [11] : 3.483334   [20] : 3.020325
[3] : 3.784733   [12] : 3.271890   [21] : 3.220219
[4] : 3.672955   [13] : 3.344085   [22] : 3.998709
[5] : 3.557408   [14] : 3.315235   [23] : 3.961214
[6] : 3.942037   [15] : 3.620479   [24] : 3.512924
[7] : 3.335304   [16] : 3.396838   [25] : 3.078143
[8] : 3.375315   [17] : 3.238764
[9] : 3.432931   [18] : 3.403484

UNKNOWN
```

Potassco

Prediction with *claspfolio*

```
$ claspfolio queens500 --decisionvalues

PRESOLVING
Reading from queens500
Solving...

Portfolio Decision Values:
[1] : 3.437538   [10] : 3.639444   [19] : 3.726391
[2] : 3.501728   [11] : 3.483334   [20] : 3.020325
[3] : 3.784733   [12] : 3.271890   [21] : 3.220219
[4] : 3.672955   [13] : 3.344085   [22] : 3.998709
[5] : 3.557408   [14] : 3.315235   [23] : 3.961214
[6] : 3.942037   [15] : 3.620479   [24] : 3.512924
[7] : 3.335304   [16] : 3.396838   [25] : 3.078143
[8] : 3.375315   [17] : 3.238764
[9] : 3.432931   [18] : 3.403484

UNKNOWN
```

Potassco

Prediction with *claspfolio*

```
$ claspfolio queens500 --decisionvalues

PRESOLVING
Reading from queens500
Solving...

Portfolio Decision Values:
[1] : 3.437538   [10] : 3.639444   [19] : 3.726391
[2] : 3.501728   [11] : 3.483334   [20] : 3.020325
[3] : 3.784733   [12] : 3.271890   [21] : 3.220219
[4] : 3.672955   [13] : 3.344085   [22] : 3.998709
[5] : 3.557408   [14] : 3.315235   [23] : 3.961214
[6] : 3.942037   [15] : 3.620479   [24] : 3.512924
[7] : 3.335304   [16] : 3.396838   [25] : 3.078143
[8] : 3.375315   [17] : 3.238764
[9] : 3.432931   [18] : 3.403484

UNKNOWN
```

Potassco

Solving with *claspfolio* (slightly verbosely)

```
$ claspfolio queens500 --quiet --autoverbose=1

PRESOLVING
Reading from queens500
Solving...

Chosen configuration: [20]
clasp --configurations=./models/portfolio.txt           \
      --modelpath=./models/                             \
      queens500 --quiet --autoverbose=1                 \
      --heu=VSIDS --sat-pre=20,25,120 --trans-ext=integ

claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models       : 1+
Time         : 4.783s (Solving: 3.96s 1st Model: 3.93s Unsat: 0.00s)
CPU Time     : 4.760s
```

Potassco

Solving with *claspfolio* (slightly verbosely)

```
$ claspfolio queens500 --quiet --autoverbose=1

PRESOLVING
Reading from queens500
Solving...

Chosen configuration: [20]
clasp --configurations=./models/portfolio.txt           \
      --modelpath=./models/                             \
      queens500 --quiet --autoverbose=1                 \
      --heu=VSIDS --sat-pre=20,25,120 --trans-ext=integ

claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models       : 1+
Time         : 4.783s (Solving: 3.96s 1st Model: 3.93s Unsat: 0.00s)
CPU Time     : 4.760s
```

Potassco

Solving with *claspfolio* (slightly verbosely)

```
$ claspfolio queens500 --quiet --autoverbose=1

PRESOLVING
Reading from queens500
Solving...

Chosen configuration: [20]
clasp --configurations=./models/portfolio.txt          \
      --modelpath=./models/                            \
      queens500 --quiet --autoverbose=1                \
      --heu=VSIDS --sat-pre=20,25,120 --trans-ext=integ

claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models       : 1+
Time         : 4.783s (Solving: 3.96s 1st Model: 3.93s Unsat: 0.00s)
CPU Time     : 4.760s
```

Potassco

Solving with *claspfolio* (slightly verbosely)

```
$ claspfolio queens500 --quiet --autoverbose=1

PRESOLVING
Reading from queens500
Solving...

Chosen configuration: [20]
clasp --configurations=./models/portfolio.txt          \
      --modelpath=./models/                            \
      queens500 --quiet --autoverbose=1                \
      --heu=VSIDS --sat-pre=20,25,120 --trans-ext=integ

claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models       : 1+
Time         : 4.783s (Solving: 3.96s 1st Model: 3.93s Unsat: 0.00s)
CPU Time     : 4.760s
```

Potassco

Solving with *claspfolio* (slightly verbosely)

```
$ claspfolio queens500 --quiet --autoverbose=1

PRESOLVING
Reading from queens500
Solving...

Chosen configuration: [20]
clasp --configurations=./models/portfolio.txt           \
      --modelpath=./models/                             \
      queens500 --quiet --autoverbose=1                 \
      --heu=VSIDS --sat-pre=20,25,120 --trans-ext=integ

claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models       : 1+
Time         : 4.783s (Solving: 3.96s 1st Model: 3.93s Unsat: 0.00s)
CPU Time     : 4.760s
```

Potassco

Outline

Potassco

clavis

- Analysis and visualization toolchain for clasp
- *clavis*
  - Event logger integrated in clasp
  - Records CDCL events like propagation, conflicts, restarts, ...
  - Generated logfiles readable with different backends
  - Easily configurable
  - Applicable to clasp variants like hclasp
- *insight*
  - Visualization backend for clavis
  - Combines information about problem structure and solving process
  - Networks for structural and aggregated information
  - Plots for temporal information and navigation

Potassco

clavis

- Analysis and visualization toolchain for clasp
- *clavis*
  - Event logger integrated in clasp
  - Records CDCL events like propagation, conflicts, restarts, ...
  - Generated logfiles readable with different backends
  - Easily configurable
  - Applicable to clasp variants like hclasp
- *insight*
  - Visualization backend for clavis
  - Combines information about problem structure and solving process
  - Networks for structural and aggregated information
  - Plots for temporal information and navigation

Potassco

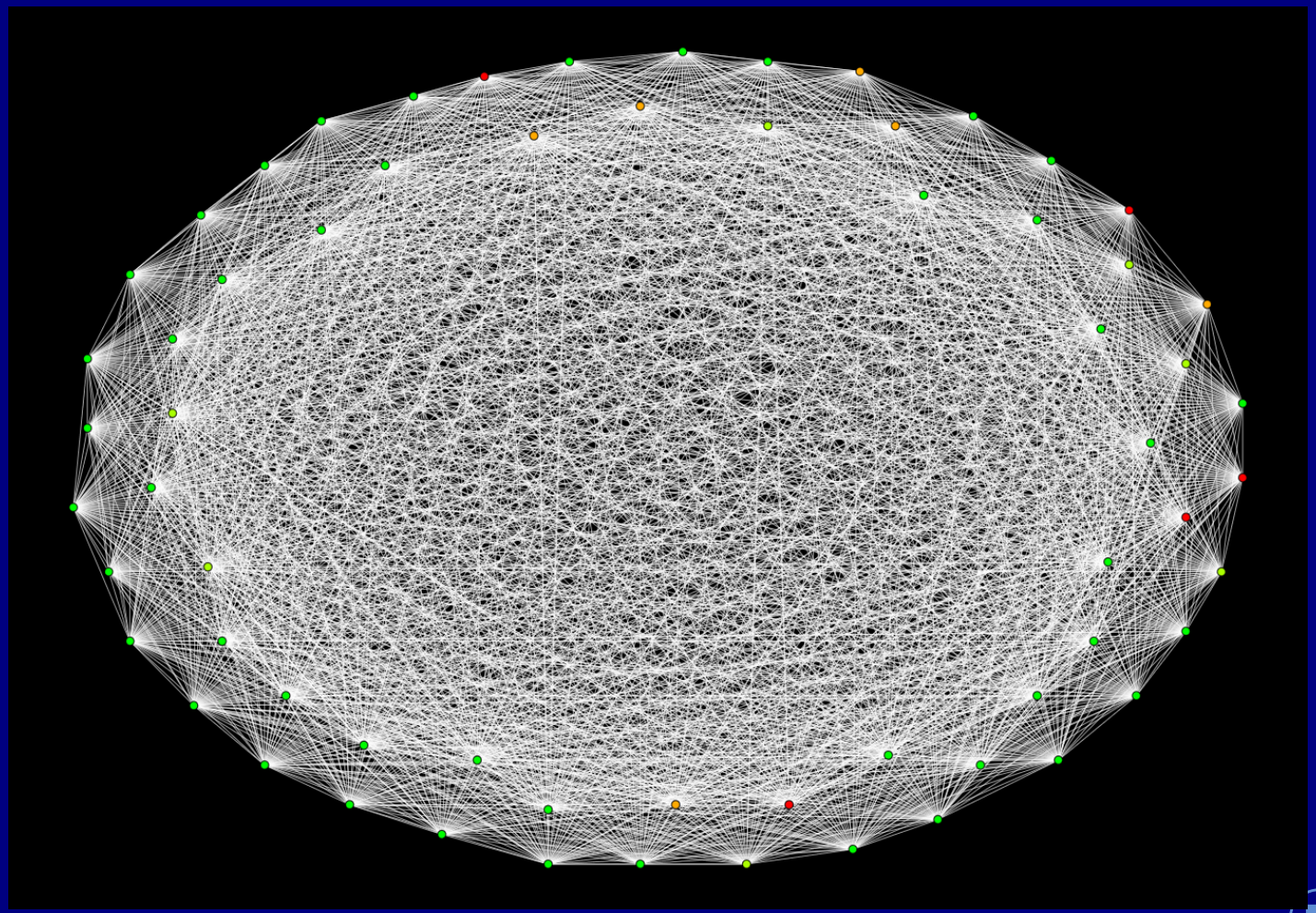
clavis

- Analysis and visualization toolchain for clasp
- *clavis*
  - Event logger integrated in clasp
  - Records CDCL events like propagation, conflicts, restarts, ...
  - Generated logfiles readable with different backends
  - Easily configurable
  - Applicable to clasp variants like hclasp
- *insight*
  - Visualization backend for clavis
  - Combines information about problem structure and solving process
  - Networks for structural and aggregated information
  - Plots for temporal information and navigation

Potassco

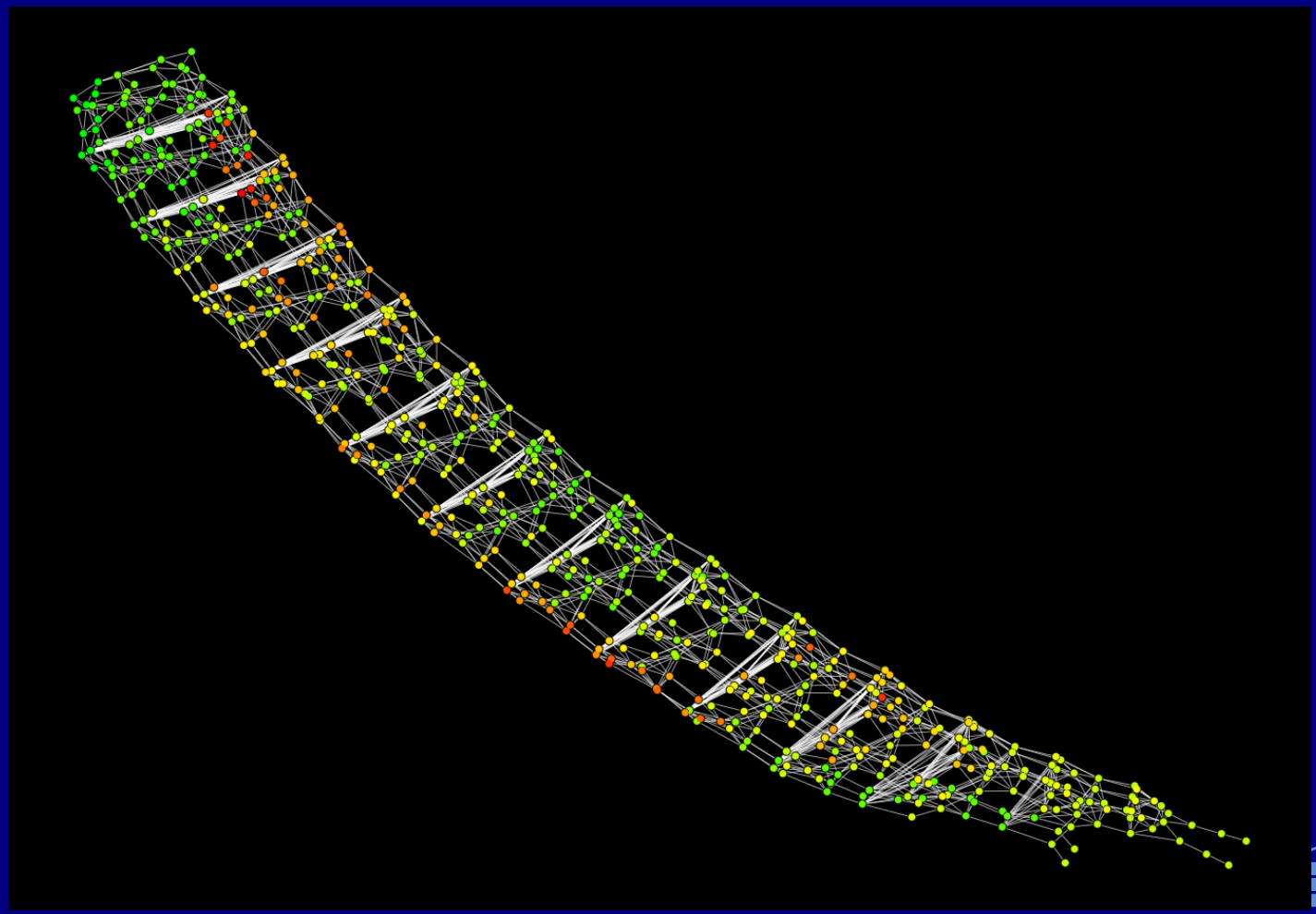
Visualization Examples

8-Queens: program interaction graph



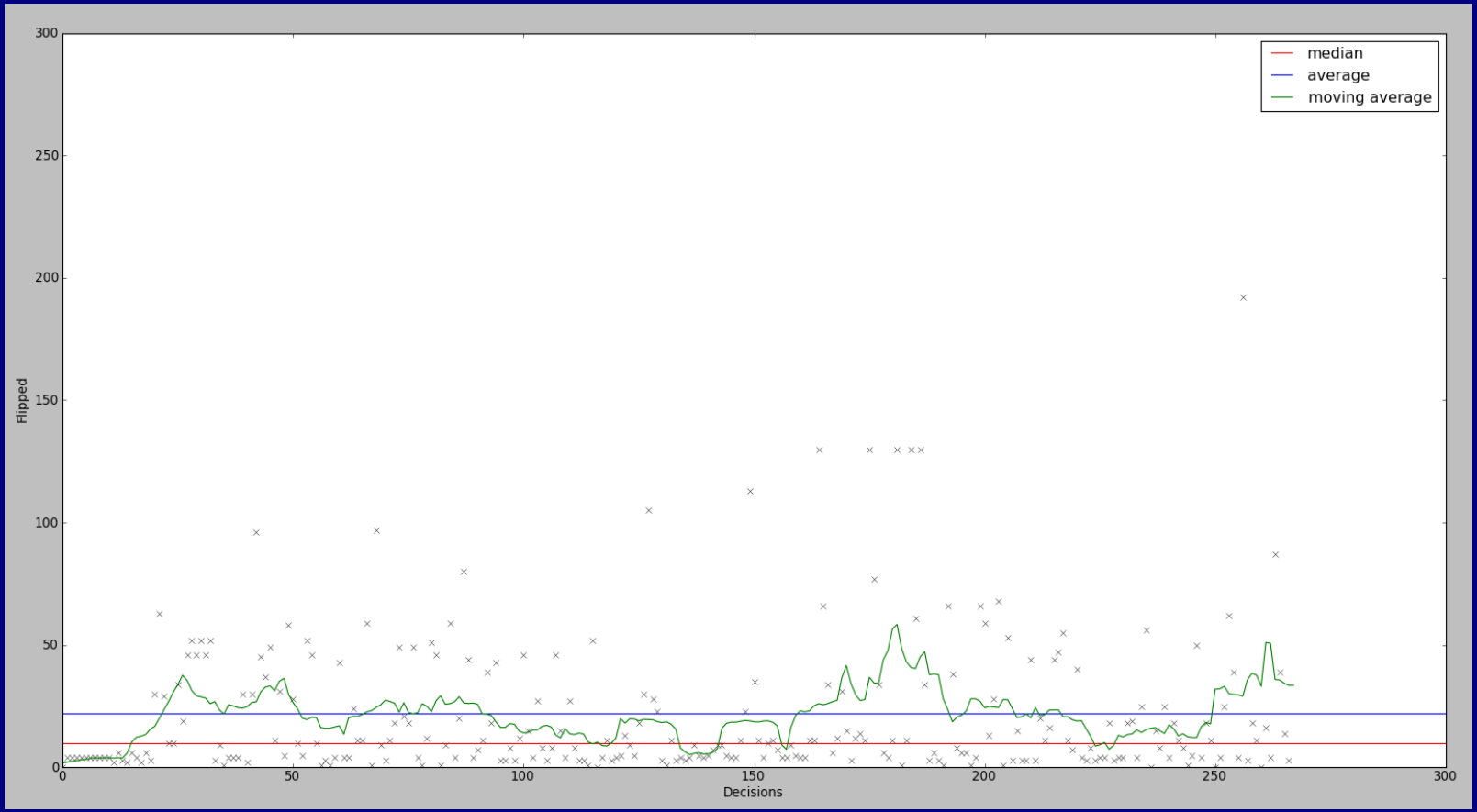
Visualization Examples

Towers of Hanoi: program interaction graph
Colors showing flipped assignments



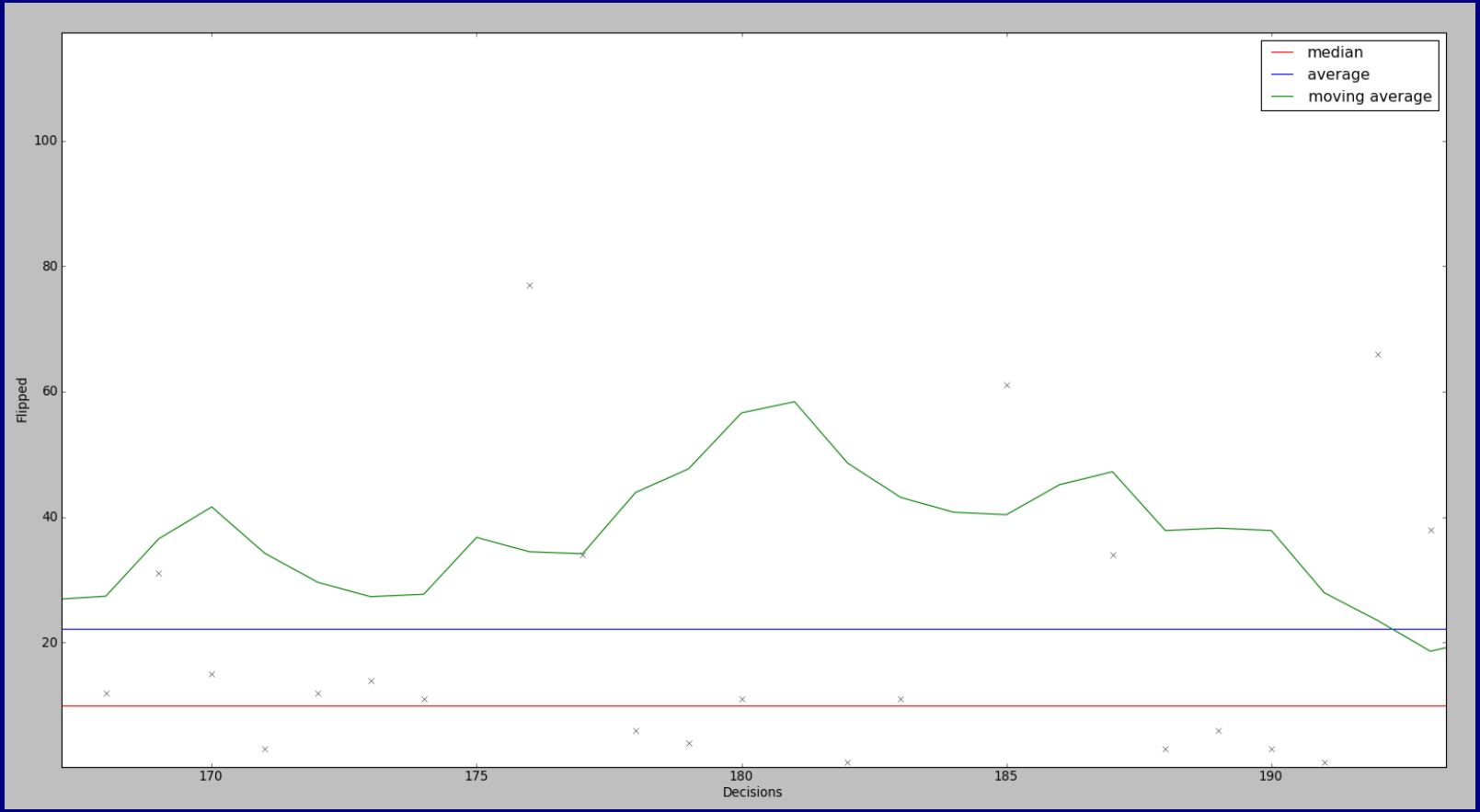
Visualization Examples

Towers of Hanoi: flipped assignments between decisions



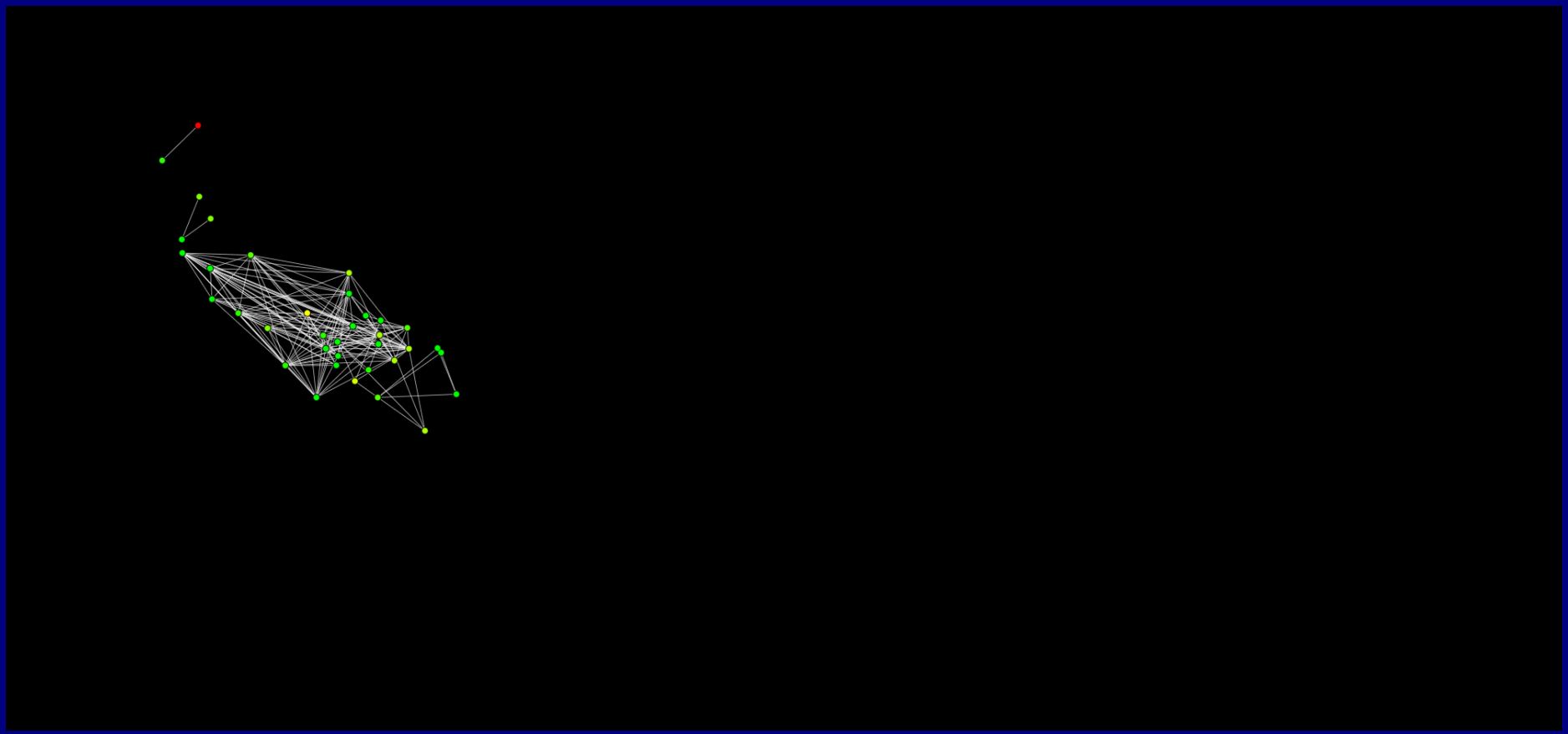
Visualization Examples

Towers of Hanoi: flipped assignments between decisions (zoomed in)



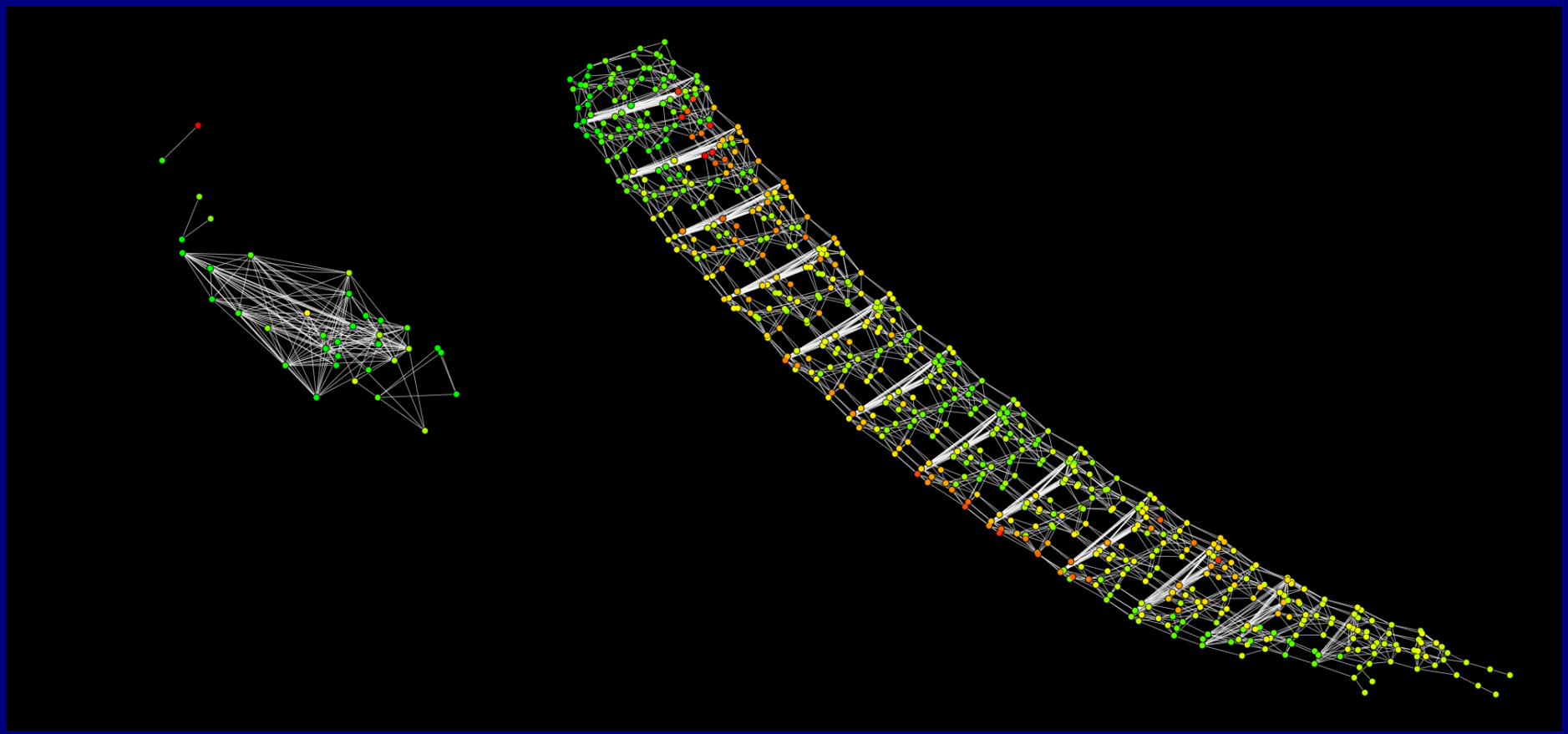
Visualization Examples

Towers of Hanoi: learned nogoods during zoomed in segment projected onto program interaction graph layout

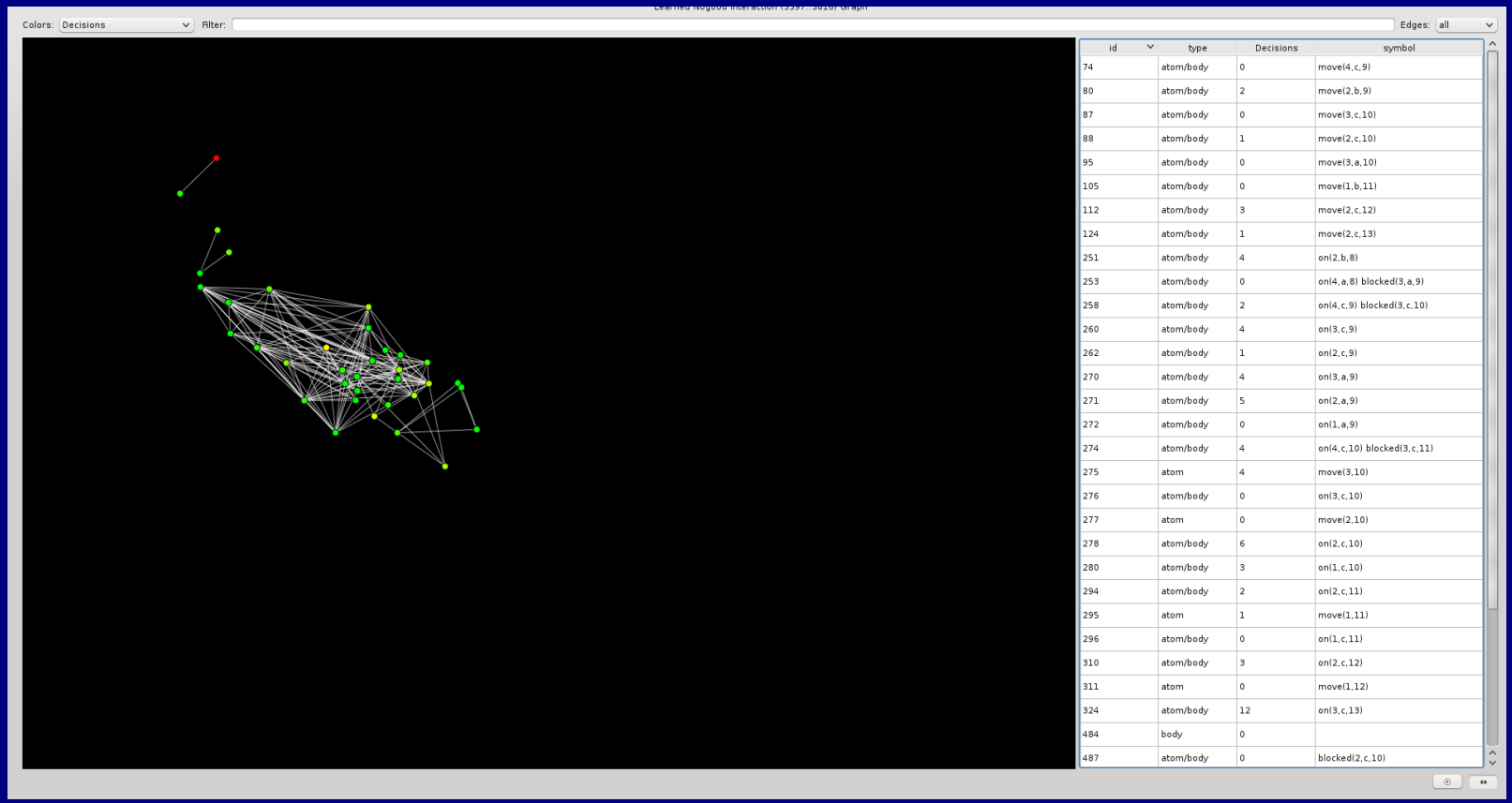


Visualization Examples

Towers of Hanoi: learned nogoods during zoomed in segment compared to program interaction graph



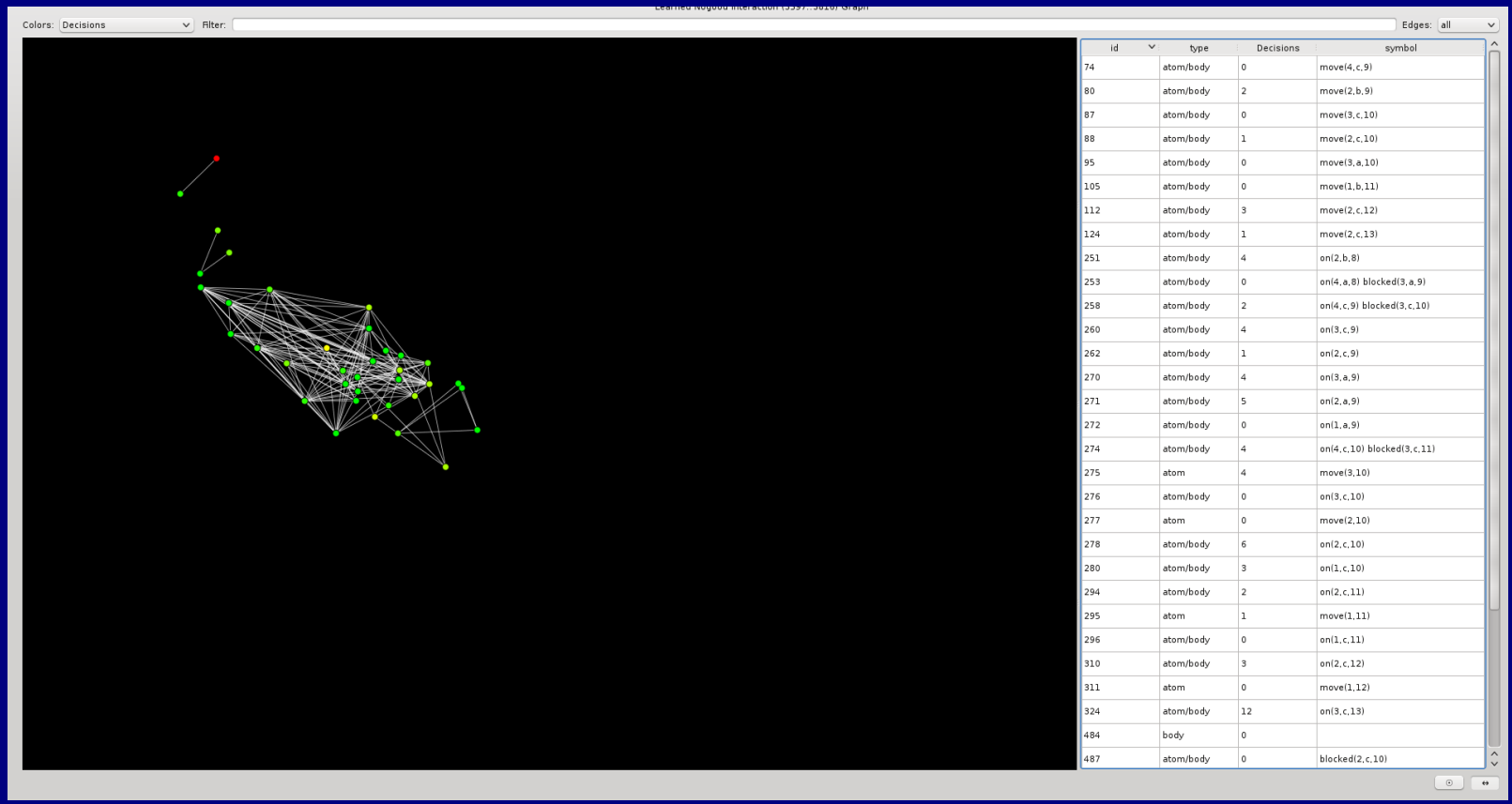
Interactive View



- Symbol table shows additional information about variables
- Search bar and symbol table allow for dynamic change of the view

Potassco

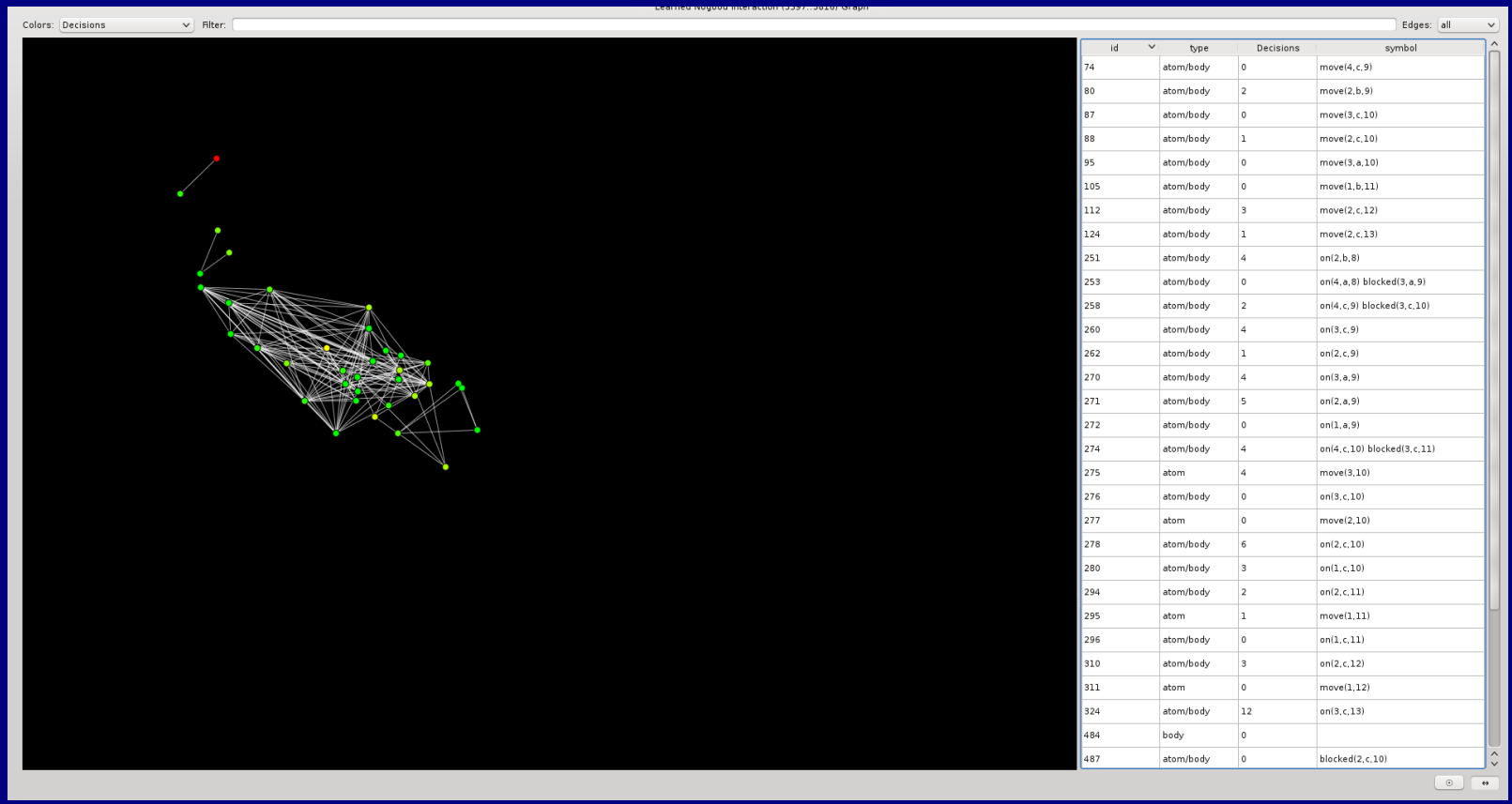
Interactive View



- Symbol table shows additional information about variables
- Search bar and symbol table allow for dynamic change of the view

Potassco

Interactive View



- Symbol table shows additional information about variables
- Search bar and symbol table allow for dynamic change of the view

Potassco

[1] C. Anger, M. Gebser, T. Linke, A. Neumann, and T. Schaub.
The nomore++ approach to answer set solving.
In G. Sutcliffe and A. Voronkov, editors, *Proceedings of the Twelfth International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)*, volume 3835 of *Lecture Notes in Artificial Intelligence*, pages 95–109. Springer-Verlag, 2005.

[2] C. Anger, K. Konczak, T. Linke, and T. Schaub.
A glimpse of answer set programming.
Künstliche Intelligenz, 19(1):12–17, 2005.

[3] Y. Babovich and V. Lifschitz.
Computing answer sets using program completion.
Unpublished draft, 2003.

[4] C. Baral.
Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge University Press, 2003.

Potassco

[5] C. Baral, G. Brewka, and J. Schlipf, editors.
Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07), volume 4483 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2007.

[6] C. Baral and M. Gelfond.
Logic programming and knowledge representation.
Journal of Logic Programming, 12:1–80, 1994.

[7] S. Baselice, P. Bonatti, and M. Gelfond.
Towards an integration of answer set and constraint solving.
In M. Gabbrielli and G. Gupta, editors, *Proceedings of the Twenty-first International Conference on Logic Programming (ICLP'05)*, volume 3668 of *Lecture Notes in Computer Science*, pages 52–66. Springer-Verlag, 2005.

[8] A. Biere.
Adaptive restart strategies for conflict driven SAT solvers.

Potassco

In H. Kleine Büning and X. Zhao, editors, *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 of *Lecture Notes in Computer Science*, pages 28–33. Springer-Verlag, 2008.

[9] A. Biere.
PicoSAT essentials.
Journal on Satisfiability, Boolean Modeling and Computation, 4:75–97, 2008.

[10] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors.
Handbook of Satisfiability, volume 185 of *Frontiers in Artificial Intelligence and Applications*.
IOS Press, 2009.

[11] G. Brewka, T. Eiter, and M. Truszczyński.
Answer set programming at a glance.
Communications of the ACM, 54(12):92–103, 2011.

[12] K. Clark.

Potassco

Negation as failure.
In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[13] M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors.
Handbook of Tableau Methods.
Kluwer Academic Publishers, 1999.

[14] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov.
Complexity and expressive power of logic programming.
In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity (CCC'97)*, pages 82–101. IEEE Computer Society Press, 1997.

[15] M. Davis, G. Logemann, and D. Loveland.
A machine program for theorem-proving.
Communications of the ACM, 5:394–397, 1962.

[16] M. Davis and H. Putnam.
A computing procedure for quantification theory.

Potassco

Journal of the ACM, 7:201–215, 1960.

[17] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, and T. Schaub.
Conflict-driven disjunctive answer set solving.
In G. Brewka and J. Lang, editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 422–432. AAAI Press, 2008.

[18] C. Drescher, M. Gebser, B. Kaufmann, and T. Schaub.
Heuristics in conflict resolution.
In M. Pagnucco and M. Thielscher, editors, *Proceedings of the Twelfth International Workshop on Nonmonotonic Reasoning (NMR'08)*, number UNSW-CSE-TR-0819 in School of Computer Science and Engineering, The University of New South Wales, Technical Report Series, pages 141–149, 2008.

[19] N. Eén and N. Sörensson.
An extensible SAT-solver.

Potassco

In E. Giunchiglia and A. Tacchella, editors, *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer-Verlag, 2004.

[20] T. Eiter and G. Gottlob.
On the computational cost of disjunctive logic programming: Propositional case.
Annals of Mathematics and Artificial Intelligence, 15(3-4):289–323, 1995.

[21] T. Eiter, G. Ianni, and T. Krennwallner.
Answer Set Programming: A Primer.
In S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M. Rousset, and R. Schmidt, editors, *Fifth International Reasoning Web Summer School (RW'09)*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer-Verlag, 2009.

[22] F. Fages.
Consistency of Clark's completion and the existence of stable models.

Journal of Methods of Logic in Computer Science, 1:51–60, 1994.

[23] P. Ferraris.
Answer sets for propositional theories.
In C. Baral, G. Greco, N. Leone, and G. Terracina, editors, *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, volume 3662 of *Lecture Notes in Artificial Intelligence*, pages 119–131. Springer-Verlag, 2005.

[24] P. Ferraris and V. Lifschitz.
Mathematical foundations of answer set programming.
In S. Artëmov, H. Barringer, A. d'Avila Garcez, L. Lamb, and J. Woods, editors, *We Will Show Them! Essays in Honour of Dov Gabbay*, volume 1, pages 615–664. College Publications, 2005.

[25] M. Fitting.
A Kripke-Kleene semantics for logic programs.
Journal of Logic Programming, 2(4):295–312, 1985.

Potassco

[26] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`.

[27] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
Engineering an incremental ASP solver.
In M. Garcia de la Banda and E. Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*, pages 190–205. Springer-Verlag, 2008.

[28] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
On the implementation of weight constraint rules in conflict-driven ASP solvers.
In Hill and Warren [44], pages 250–264.

[29] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
Answer Set Solving in Practice.

Potassco

Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

[30] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
clasp: A conflict-driven answer set solver.
In Baral et al. [5], pages 260–265.

[31] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set enumeration.
In Baral et al. [5], pages 136–148.

[32] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set solving.
In Veloso [68], pages 386–392.

[33] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Advanced preprocessing for answer set solving.
In M. Ghallab, C. Spyropoulos, N. Fakotakis, and N. Avouris, editors, *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08)*, pages 15–19. IOS Press, 2008.

Potassco

[34] M. Gebser, B. Kaufmann, and T. Schaub.
The conflict-driven answer set solver clasp: Progress report.
In E. Erdem, F. Lin, and T. Schaub, editors, *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Artificial Intelligence*, pages 509–514. Springer-Verlag, 2009.

[35] M. Gebser, B. Kaufmann, and T. Schaub.
Solution enumeration for projected Boolean search problems.
In W. van Hoeve and J. Hooker, editors, *Proceedings of the Sixth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09)*, volume 5547 of *Lecture Notes in Computer Science*, pages 71–86. Springer-Verlag, 2009.

[36] M. Gebser, M. Ostrowski, and T. Schaub.
Constraint answer set solving.
In Hill and Warren [44], pages 235–249.

Potassco

[37] M. Gebser and T. Schaub.
Tableau calculi for answer set programming.
In S. Etalle and M. Truszczyński, editors, *Proceedings of the Twenty-second International Conference on Logic Programming (ICLP'06)*, volume 4079 of *Lecture Notes in Computer Science*, pages 11–25. Springer-Verlag, 2006.

[38] M. Gebser and T. Schaub.
Generic tableaux for answer set programming.
In V. Dahl and I. Niemelä, editors, *Proceedings of the Twenty-third International Conference on Logic Programming (ICLP'07)*, volume 4670 of *Lecture Notes in Computer Science*, pages 119–133. Springer-Verlag, 2007.

[39] M. Gelfond.
Answer sets.
In V. Lifschitz, F. van Harmelen, and B. Porter, editors, *Handbook of Knowledge Representation*, chapter 7, pages 285–316. Elsevier Science, 2008.

Potassco

[40] M. Gelfond and N. Leone.
Logic programming and knowledge representation — the A-Prolog perspective.
Artificial Intelligence, 138(1-2):3–38, 2002.

[41] M. Gelfond and V. Lifschitz.
The stable model semantics for logic programming.
In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. MIT Press, 1988.

[42] M. Gelfond and V. Lifschitz.
Logic programs with classical negation.
In D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming (ICLP'90)*, pages 579–597. MIT Press, 1990.

[43] E. Giunchiglia, Y. Lierler, and M. Maratea.
Answer set programming based on propositional satisfiability.

Potassco

Journal of Automated Reasoning, 36(4):345–377, 2006.

[44] P. Hill and D. Warren, editors.
Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09), volume 5649 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[45] J. Huang.
The effect of restarts on the efficiency of clause learning.
In Veloso [68], pages 2318–2323.

[46] K. Konczak, T. Linke, and T. Schaub.
Graphs and colorings for answer set programming.
Theory and Practice of Logic Programming, 6(1-2):61–106, 2006.

[47] J. Lee.
A model-theoretic counterpart of loop formulas.
In L. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 503–508. Professional Book Center, 2005.

Potassco

[48] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello.
The DLV system for knowledge representation and reasoning.
ACM Transactions on Computational Logic, 7(3):499–562, 2006.

[49] V. Lifschitz.
Answer set programming and plan generation.
Artificial Intelligence, 138(1-2):39–54, 2002.

[50] V. Lifschitz.
Introduction to answer set programming.
Unpublished draft, 2004.

[51] V. Lifschitz and A. Razborov.
Why are there so many loop formulas?
ACM Transactions on Computational Logic, 7(2):261–268, 2006.

[52] F. Lin and Y. Zhao.
ASSAT: computing answer sets of a logic program by SAT solvers.
Artificial Intelligence, 157(1-2):115–137, 2004.

Potassco

[53] V. Marek and M. Truszczyński.
Nonmonotonic logic: context-dependent reasoning.
Artifical Intelligence. Springer-Verlag, 1993.

[54] V. Marek and M. Truszczyński.
Stable models and an alternative logic programming paradigm.
In K. Apt, V. Marek, M. Truszczyński, and D. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398.
Springer-Verlag, 1999.

[55] J. Marques-Silva, I. Lynce, and S. Malik.
Conflict-driven clause learning SAT solvers.
In Biere et al. [10], chapter 4, pages 131–153.

[56] J. Marques-Silva and K. Sakallah.
GRASP: A search algorithm for propositional satisfiability.
IEEE Transactions on Computers, 48(5):506–521, 1999.

[57] V. Mellarkod and M. Gelfond.
Integrating answer set reasoning with constraint solving techniques.

In J. Garrigue and M. Hermenegildo, editors, *Proceedings of the Ninth International Symposium on Functional and Logic Programming (FLOPS'08)*, volume 4989 of *Lecture Notes in Computer Science*, pages 15–31. Springer-Verlag, 2008.

[58] V. Mellarkod, M. Gelfond, and Y. Zhang.
Integrating answer set programming and constraint logic programming.
Annals of Mathematics and Artificial Intelligence, 53(1-4):251–287, 2008.

[59] D. Mitchell.
A SAT solver primer.
Bulletin of the European Association for Theoretical Computer Science, 85:112–133, 2005.

[60] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik.
Chaff: Engineering an efficient SAT solver.
In *Proceedings of the Thirty-eighth Conference on Design Automation (DAC'01)*, pages 530–535. ACM Press, 2001.

Potassco

[61] I. Niemelä.
Logic programs with stable model semantics as a constraint programming paradigm.
Annals of Mathematics and Artificial Intelligence, 25(3-4):241–273, 1999.

[62] R. Nieuwenhuis, A. Oliveras, and C. Tinelli.
Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T).
Journal of the ACM, 53(6):937–977, 2006.

[63] K. Pipatsrisawat and A. Darwiche.
A lightweight component caching scheme for satisfiability solvers.
In J. Marques-Silva and K. Sakallah, editors, *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer-Verlag, 2007.

[64] L. Ryan.

Potassco

Efficient algorithms for clause-learning SAT solvers.
Master's thesis, Simon Fraser University, 2004.

[65] P. Simons, I. Niemelä, and T. Soininen.
Extending and implementing the stable model semantics.
Artificial Intelligence, 138(1-2):181–234, 2002.

[66] T. Syrjänen.
Lparse 1.0 user's manual.

[67] A. Van Gelder, K. Ross, and J. Schlipf.
The well-founded semantics for general logic programs.
Journal of the ACM, 38(3):620–650, 1991.

[68] M. Veloso, editor.
Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07). AAAI/MIT Press, 2007.

[69] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik.
Efficient conflict driven learning in a Boolean satisfiability solver.

Potassco

In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285. ACM Press, 2001.